Parallele adaptive Schwarmsuche für Blackbox-Probleme

Von der Fakultät für Elektrotechnik und Informatik

der Gottfried Wilhelm Leibniz Universität Hannover

zur Erlangung des akademischen Grades

D O K T O R - I N G E N I E U R

(abgekürzt: Dr.-Ing.)

genehmigte Dissertation

von Herrn

M.Sc. Romeo Shuka

geboren am 20. Oktober 1985

in Durres, Albanien

Hannover – 2019

Referent:	Prof. Dr.-Ing. habil. Jürgen Brehm
Korreferent:	Prof. Dr. Michael Rohs
Tag der Promotion:	23. September 2019

Kurzfassung

Schlüsselwörter — Blackbox, Optimierung, Particle Swarm Optimization, Space Mission, parallel, adaptiv

In der heutigen Wissenschaft und Wirtschaft haben wir es oft mit Systemen zu tun, welche aus Problemen bestehen, die sehr komplex und nicht einfach zu lösen sind. Aufgrund der zunehmenden Komplexität und der teilweise fehlenden Informationen ist es bereits heutzutage nicht mehr möglich, solche Probleme – welche als Blackbox-Probleme klassifiziert werden – per Hand zu lösen. Um das Maximum oder Minimum einer Funktion zu finden, wird auf Optimierungsmethoden zurückgegriffen, die uns ermöglichen, eine optimale Lösung für das Problem zu suchen und ggf. zu finden. Stochastische Methoden haben in den letzten Jahren gezeigt, dass sie sehr gut geeignet sind, solche Probleme zu lösen. Der Vorteil der Verwendung von stochastischen Methoden ist, dass sie nicht den Gradienten des zu optimierenden Problems verwenden, so dass sie sowohl bei großen als auch bei komplexen Optimierungsproblemen erfolgreich angewendet werden können. Diese Vielseitigkeit hat aber ihren Preis. Es gibt hauptsächlich drei wesentliche Aspekte, die die Effizienz der Lösung beeinträchtigen:

1. Die realen Probleme werden immer größer und komplizierter, was erhebliche Ressourcen in Zeit und Hardware erfordert.

2. Optimierungsprobleme sind durch mehrere lokale Optima charakterisiert, die ein Verfahren zur Vermeidung einer zu frühen Konvergenz erfordern.

3. Algorithmen erfordern einige problembedingte Anpassungen ihrer Verhaltensparameter, um bessere Ergebnisse zu erzielen.

In dieser Arbeit wird ein Framework (Parallel Adaptive Swarm Search - PASS) vorgestellt, das die Ermittlung der möglichst optimalen Lösung für Black-box Probleme gewährleistet. Durch das Framework kann der Nutzer ein Mapping des Algorithmus an die vorhandene Hardware und Software (Problemart) realisieren. Das Problem der Parallelisierung wird durch die Umwandlung des Algorithmus von seriell zu parallel gelöst. Das Problem des Stillstands wird durch das Benutzen des Island Models behandelt und für das Problem der Adaptivität wird ein neuer Suchalgorithmus vorgestellt, welcher die Suche der optimalen Parameter beschleunigt. In dieser Arbeit wird PASS mit bekannten Verfahren aus der Literatur (wie z.B. Particle Swarm Optimization, Differential Evolution, Artificial Bee Colony usw.) mit unterschiedlichen Benchmark-Problemen verglichen. Die erste Gruppe der Benchmark-Probleme besteht aus synthetischen Funktionen bekannt aus der Literatur (wie z.B. Rastrigin Funktion, Rosenbrock Funktion) und die zweite Gruppe besteht aus Problemen aus der realen Welt.

Untersuchungen in dieser Arbeit haben gezeigt, dass die Anpassungen zu besseren Ergebnissen führen. Durch die adaptive Natur des Frameworks, ist es in vielen Rechnerarchitekturen nutzbar und für viele Probleme anwendbar.

Abstract

Keywords — black-box, optimization, particle swarm optimization, interplanetary space mission, parallel, adaptive

In today's science and economy we often have to deal with systems that consist of problems that are very complex and not easy to solve. Due to the increasing complexity and the partly missing information it is no longer possible to solve such problems – which are classified as black box problems – by hand. In order to find the maximum or minimum, optimization methods are used that allow us to search for an optimal solution to the problem and, if necessary, to find it. In recent years, stochastic methods have shown that they are very well suited to solve such problems.

The advantage of using these methods is that it does not use the gradient of the problem to be optimized, thus, it can be successfully applied to both, large scale and complex optimization problems. This versatility comes at a price, as three major restrictions limit the solution's efficiency:

1. Real-world problems are getting larger and more complicated, which requires significant resources in time and hardware

2. Optimization problems are characterized by multiple local optima, requiring a method to avoid early stagnation

3. Depending on the problem, algorithms are in need of some adjustments to their behavioral parameters in order to achieve better results.

In this paper, we present a framework (Parallel Adaptive Swarm Search - PASS) that ensures the determination of the best possible solution for black box problems.

The framework allows the user to map the algorithm to the existing hardware and software (problem type). All three problems mentioned above are covered. The problem of parallelization is solved by converting the algorithm from serial to parallel. The problem of stagnation is solved by using the Island Model and for the problem of adaptivity a new search algorithm is introduced, which accelerates the search of optimal parameters. In this thesis, PASS is compared to known methods from literature (e.g. Particle Swarm Optimization, Differential Evolution, Artificial Bee Colony etc.) in different benchmark problems. The first group of benchmark problems consists of synthetic functions known from the literature (such as Rastrigin function, Rosenbrock function) and the second group consists of problems from the real world.

Studies in this thesis have shown that the adjustments lead to better results. Due to the adaptive nature of the framework, it is usable in many computer architectures and applicable to many problems.

Inhaltsverzeichnis

1

Einleitung

Im Grunde bewegen nur zwei Fragen die Menschheit: Wie hat alles angefangen und wie wird alles enden?

STEPHEN HAWKING

1.1 Motivation

In der heutigen Wissenschaft und Wirtschaft haben wir es oft mit Systemen zu tun, welche aus Problemen bestehen, die sehr komplex und nicht einfach zu lösen sind. Aufgrund der zunehmenden Komplexität und der teilweise fehlenden Informationen ist es bereits heutzutage nicht mehr möglich, solche Probleme – welche als *Blackbox-Probleme (BBPs)* klassifiziert werden – per Hand zu lösen. Diese Art von Problemen sind überall in der realen Welt anzutreffen, wie z. B. in der Finanzbranche [161], Medizin [157], Robotik [114] oder Elektrotechnik [82]. Egal in welcher Domäne wir uns befinden, alle haben etwas gemeinsam: Den Versuch, mit wenig Aufwand, viel Ertrag zu erzielen.

In der Finanzbranche wird angestrebt, den Profit mit minimalen Kosten zu *maximieren*, in der Medizin wird versucht, die Zeit zur Feststellung einer Diagnose zu *minimieren*, und in der Elektrotechnik wird versucht, die Kosten der Bauteile auf ein *Minimum* zu reduzieren. Wenn wir einen Blick auf die verschiedenen Domänen werfen, können wir sehr schnell feststellen, dass das Ziel in allen Bereichen das Finden eines *Maximums* oder *Minimums* ist. In mathematischer Hinsicht reden wir hier über *Optimierung*. Wir haben eine *Zielfunktion*, welche definiert, ob wir es mit einer Maximierung oder Minimierung zu tun haben. Jede Lösung dieser Funktion hat eine bestimmte Güte, welche in der Evolutionsbiologie auch *Fitnesswert* genannt wird [37]. Die Herausforderungen bei BBPs, im Vergleich zu anderen Problemklassen, liegt in der Tatsache, dass das Optimum nicht bekannt ist. Im besten Fall können existierende Lösungen – falls vorhanden – als Referenz genommen werden.

Um das Maximum oder Minimum zu finden, wird auf Optimierungsmethoden zurückgegriffen, die uns ermöglichen, eine optimale Lösung für das Problem zu suchen und ggf. zu finden. Stochastische Methoden wie Evolutionäre Algorithmen (EA) oder schwarmbasierte Algorithmen haben die letzten Jahre gezeigt, dass sie sehr gut geeignet sind, die BBPs zu lösen [111]. Dabei handelt es sich um Heuristiken, welche an Analogien aus der Natur angelehnt sind, wie z. B. PSO [43] oder Artificial Bee Colony (ABC) [83].

Der Vorteil der Verwendung von stochastischen Methoden ist, dass sie nicht den Gradienten des zu optimierenden Problems verwenden, so dass sie sowohl bei *großen* als auch bei *komplexen* Optimierungsproblemen erfolgreich angewendet

werden können. Diese Vielseitigkeit hat aber ihren Preis. Es gibt hauptsächlich drei wesentliche Aspekte, die die Effizienz der Lösung beeinträchtigen:

Parallelisierung: Die realen Probleme werden immer größer und komplizierter oder sie müssen in sehr kurzer Zeit gelöst werden, was erhebliche Ressourcen in Zeit und Hardware erfordert.

Stillstand: Optimierungsprobleme sind durch mehrere lokale Optima charakterisiert, die ein Verfahren zur Vermeidung einer zu frühen Konvergenz erfordern [24].

Adaptivität: Algorithmen erfordern einige problembedingte Anpassungen ihrer Verhaltensparameter, um bessere Ergebnisse zu erzielen [162].

Die Lösungen dieser Probleme bilden das Thema dieser Arbeit. Diese Arbeit stellt ein Verfahren und die dazugehörigen Werkzeuge zur Verfügung. Sie ist in drei Hauptteile untergliedert:

1. Analyse der vorhandenen Hardware (zur Verfügung stehende Performance) und Problemart, um festzustellen, ob eine Parallelisierung vorteilhaft ist.

2. Vermeidung der zu frühen Konvergenz aufgrund lokaler Optima. (Komplexität des benötigten Verfahrens)

3. Finden der optimalen Parameter eines bestimmten Problems. (Mapping der Anwendung auf die Architektur)

1.2 Problembeschreibung

1.2.1 Parallelisierung

Die Komplexität der Probleme, die großen Datenmengen und die Zeitanforderungen, mit denen wir es heutzutage zu tun haben, machen in vielen Fällen eine Parallelisierung dringend notwendig. Die größte Herausforderung hierbei liegt in den gegenläufigen Effekten zwischen Rechendauer und Qualität des Optimums. Eine

geringe Rechenzeit ist in Hinsicht auf die Anforderungen des Auftraggebers genauso wünschenswert wie eine möglichst gute und gleichzeitig akzeptierbare Lösung.

Die Komplexität der existierenden parallelen Architekturen erfordert ein grundlegendes Umdenken bei der Softwareentwicklung [89], um das oben genannte Ziel zu erreichen. Die erste Herausforderung liegt in der Art der zu verwendenden Parallelisierungsumgebung. Eine Möglichkeit wäre der *Nachrichtenaustausch* zwischen mehreren Maschinen. Dieses Verfahren wird meistens in *High Performance Computing Clusters (HPCCs)* verwendet und die Realisierung erfolgt mit Hilfe des *Message Passing Interface (MPI)* [108]. Eine zweite Möglichkeit für die Realisierung ist eine *Shared-Memory*-Architektur mittels *Open Multi-Processing (OpenMP)* [117]. Falls wir über mehrere *General Purpose Graphics Processing Units (GPGPUs)* verfügen, können wir die Parallelisierung auf einen Grafikprozessor auslagern. Die Realisierung hier kann beispielsweise entweder durch eine von Nvidia entwickelte Programmiertechnik namens *Compute Unified Device Architecture (CUDA)* [27] oder durch die *Open Computing Language (openCL)* [116] erfolgen. Die Methoden können entweder einzeln oder zusammen implementiert werden. Abbildung 1.1 gibt einen Überblick der genannten Methoden.

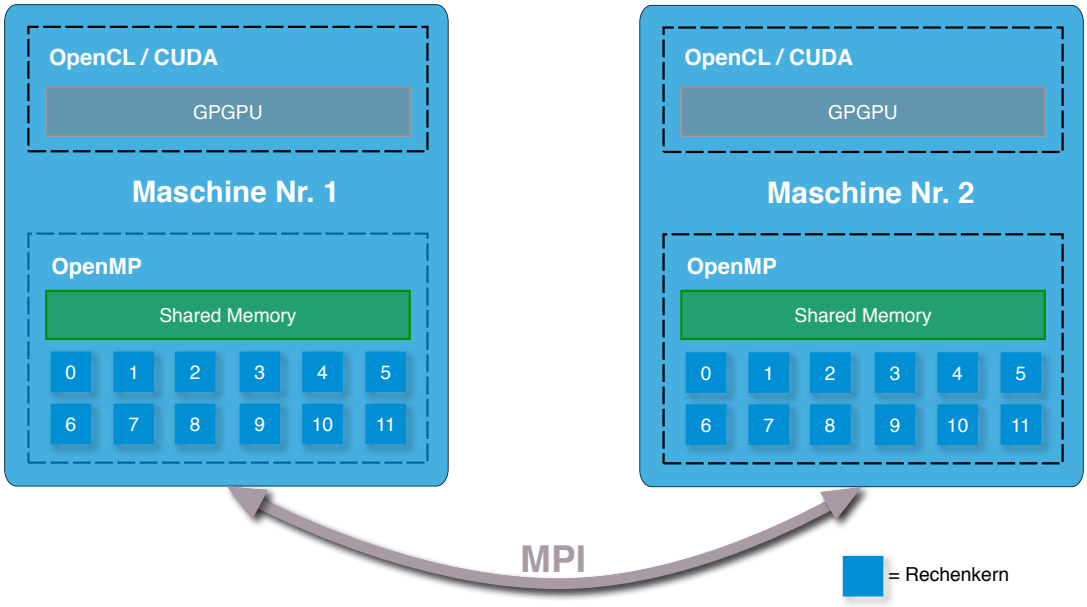


Abbildung 1.1 – Parallele Programmierumgebung. Die verschiedenen Möglichkeiten und deren Zusammenhänge in der parallelen Welt.

Die Verteilung der Rechenlast auf mehrere Prozesse oder Rechner erfordert immer einen zusätzlichen Aufwand, den sogenannten *Overhead*. Dieser kann z. B. bei der Erstellung und Synchronisierung der Threads oder beim Verschicken der Nachrichten durch das Netzwerk in einem HPCC entstehen. Unter Betrachtung des Amdahlschen Gesetzes [6] sollte ein minimaler Overhead erzielt werden. Leider ist es noch nicht möglich, eine vollautomatisierte und gleichzeitig effiziente Parallelisierung (d. h minimaler Overhead) durchzuführen. In solchen Fällen muss der Entwickler manuelle Änderungen und Transformationen im Code vornehmen, um eine effiziente parallele Ausführung des Programms zu erreichen.

1.2.2 Stillstand

Die Probleme der realen Welt bestehen aus mehreren Einflussvariablen, welche in einem bestimmten Verhältnis zueinander stehen. Dieses Verhältnis wird durch eine mathematische Funktion vorgegeben. Solche Funktionen sind in der Regel sehr komplex und bestehen aus mindestens je einem lokalen und globalen Extremwert (siehe Abb. 1.2).

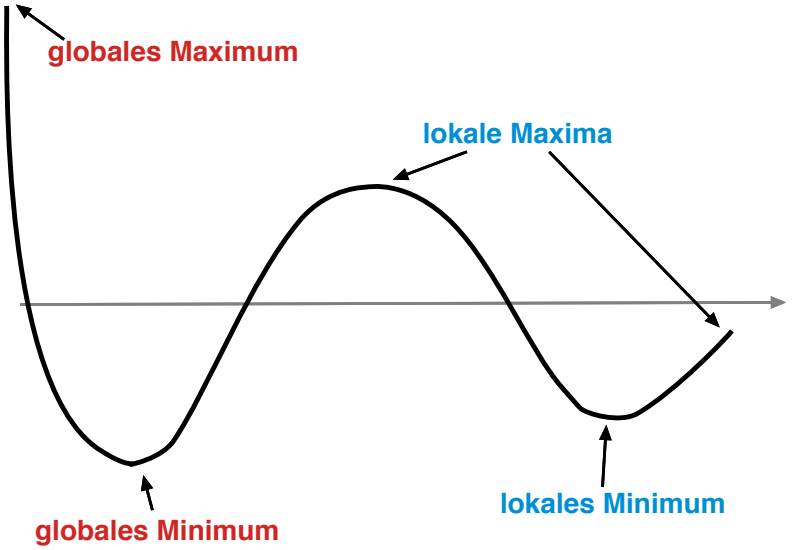


Abbildung 1.2 – Extremwerte. Minima und Maxima einer Funktion.

Eines der größten Probleme der stochastischen Optimierungsmethoden liegt an der zu frühen Konvergenz zu den lokalen Minima bzw. Maxima [24], welche zu

einem „*Stillstand*“ bei der Suche des Optimums führt. Dieses Fehlverhalten lässt sich nicht total vermeiden, deshalb wird versucht, es so weit wie möglich zu unterbinden. In der Forschung werden hierfür verschiedene Lösungen vorgeschlagen. Diese reichen von relativ einfachen Methoden wie z. B. einem Neustart des Algorithmus [50] bis hin zu komplexeren Methoden wie dem Island Modell [97].

Jede Methode hat ihre Vor- und Nachteile und ist nur für bestimmte Anwendungsfälle geeignet. Deshalb ist die korrekte Wahl, welche Methode für einen Algorithmus und eine Problemart benutzt wird, sehr entscheidend und nicht einfach zu lösen. Die genauere Implementierung und Konfiguration des Verfahrens stellt eine weitere Problematik dar, welche auch gelöst werden muss.

1.2.3 Adaptivität

Die meisten Algorithmen arbeiten mit verschiedenen konfigurierbaren Parametern. Die Anzahl der Parameter variiert von Algorithmus zu Algorithmus. Das Setzen dieser Parameter hat einen großen Einfluss auf das Verhalten des Algorithmus [146] und ist somit ausschlaggebend für die Qualität der Lösung. Die Wahl des besten Parameters ist meist zeitaufwändig und nicht einfach zu erledigen. Die existierenden Methoden dauern entweder zu lange (z. B. die Brute-Force-Methode) [25] oder verschieben das Parametersuche-Problem auf eine höhere Ebene (z. B. die Meta-Optimierung) [48], daher muss eine Lösung gefunden werden, welche deutlich schneller ist und gute bis sehr gute Ergebnisse liefert.

1.3 Beitrag zur Forschung

Die vorliegende Dissertation beschäftigt sich mit den in diesem Kapitel beschriebenen Problemen und nutzt unter anderem Techniken und Methoden aus der Forschung, um diese zu lösen. Hierzu wurde das Framework *PASS* entwickelt. PASS unterstützt sowohl den Entwickler als auch den Endbenutzer. Aus Entwicklersicht bietet das Framework die Möglichkeit, es mit anderen Algorithmen oder Problemen zu erweitern. Die dafür benötigten Schnittstellen und die dazugehörige Dokumentation stehen zur Verfügung. Als Endbenutzer kann man die Software einfach anwenden, ohne große Kenntnisse in dem Bereich der Optimierung zu ha-

ben. Die automatische Anpassung und Aktivierung bzw. Deaktivierung bestimmter Funktionen macht PASS sehr benutzerfreundlich und einfach zu bedienen.

Es werden von PASS alle drei (in Abschnitt 1.2) beschriebenen Probleme abgedeckt. Das Framework ist sowohl für den privaten Rechner als auch für HPCCs konzipiert. Am Anfang findet eine automatische Hardwareanalyse statt, um herauszufinden, ob die zur Verfügung stehende Hardware für Parallelisierung geeignet ist. Hier wird durch das maschinelle Lernen ein Modell gebildet, um eine Speedup-Vorhersage zu erstellen. Nachdem entschieden wird, ob die Hardware für die Parallelisierung geeignet ist, wird eine Parameteruntersuchung durchgeführt. Dabei werden verschiedene Parameterkonfigurationen untersucht und diejenigen vorgeschlagen, die die besten Ergebnisse liefern. Dieser Schritt erfolgt ebenso automatisch und braucht keine Angaben von außen. Die ersten oben genannten zwei Schritte dienen dazu, den Algorithmus für das zu untersuchende Problem und die vorhandene Hardware bestmöglich zu konfigurieren. Weiterhin besteht die Möglichkeit, den Algorithmus mittels MPI parallel auf verschiedenen Rechnerknoten laufen zu lassen, um die Genauigkeit der Lösung zu erhöhen.

Den Schwerpunkt der untersuchten Algorithmen bildet der Particle Swarm Optimization (PSO) Algorithmus. Dieser Algorithmus wurde exemplarisch ausgewählt, da er für viele Probleme ein sehr gutes Konvergenzverahlten zeigt und sich auch für eine Parallelisierung eignet. Obwohl die Methoden dieser Arbeit anhand dieses Algorithmus beschrieben werden, sind die beschriebenen Techniken auf andere Algorithmenklassen erweiterbar, was in Abschnitt 3 diskutiert wird.

1.4 Aufbau der Arbeit

Die hier vorliegende Arbeit ist in folgende Kapitel gegliedert: Kapitel 2 gibt einen Überblick über die Welt der Optimierung. Kapitel 3 diskutiert den Stand der Technik und die Grundlagen, welche dazu dienen den Umfang dieser Arbeit zu definieren. Hier werden auch die verwandten Arbeiten vorgestellt und diskutiert. Im Kapitel 4 wird die Systemarchitektur von PASS vorgestellt und im darauffolgenden Kapitel 5 wird auf die Details eingegangen. Die Evaluationsergebnisse der Einzelteile und des gesamten Frameworks werden in Kapitel 5 und 6 vorgestellt und diskutiert. Dort wird PASS u. a. mit anderen aktuellen Algorithmen verglichen. Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick in Kapitel 7 ab.

2

Grundlagen

Optimierung

Ich kann freilich nicht sagen, ob es besser werden wird, wenn es anders wird; aber so viel kann ich sagen: Es muss anders werden, wenn es gut werden soll.

Georg Christoph Lichtenberg

Das nachfolgende Kapitel gibt einen Überblick über das Thema Optimierung. Grundlegende Definitionen werden im Abschnitt 2.1 bereitgestellt. In den Abschnitten 2.1.1, 2.1.2, 2.1.3 und 2.2 werden die verschiedenen Klassifizierungen der Optimierungsprobleme vorgestellt. In dem darauffolgenden Abschnitt 2.3 werden die Optimierungsprobleme, mit denen sich diese Arbeit beschäftigt, aufgelistet und erklärt. Zum Schluss wird ein Überblick gegeben und die wichtigsten Definitionen werden zusammengefasst.

2.1 Optimierungsproblem

Unter dem Begriff der Optimierung wird im Allgemeinen das Suchen einer Maßnahme zur Verbesserung des aktuellen Zustands verstanden. In der Mathematik versteht man unter Optimierung das Aufsuchen eines Extremwertes (siehe Def. 2.1.1).

Definition 2.1.1 (Extremwert) *Ein Extremwert ist der Oberbegriff für ein lokales bzw. globales Maximum oder Minimum. Ein lokales Minimum ist der Wert einer Funktion, in deren Umgebung keine kleineren Werte zu finden sind. Analog gilt die Definition vom lokalen Maximum.*

Wie im Kapitel 1.1 erwähnt, ist die Optimierung in vielen Bereichen von Industrie und Wirtschaft vertreten. Einige Beispiele sind:

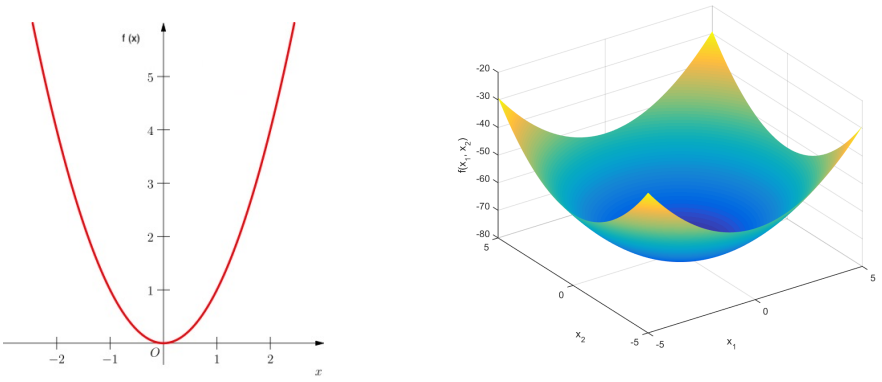
- **Finanzen**: Z. B. Portfolio-Optimierung [47], wo eine Maximierung des Gewinns erzielt wird, indem man die optimale Verteilung von Investitionen plant.
- **Flugzeuge**: Z. B. die Reduktion von Fluglärm in der Nähe eines Flughafens [91].
- **Klima**: Z. B. Entwicklung von CO_2-armen Energietechnologien [149].

Die mathematische Formulierung der oben genannten Probleme führt auf ein Optimierungsproblem. Anhand ihrer Fitnesslandschaft (siehe Def. 2.1.2) wird zwischen *statischen* und *dynamischen* Optimierungsproblemen unterschieden. Diese werden wie folgt definiert [65]:

- *Statisches Optimierungsproblem:* Minimierung oder Maximierung einer Funktion mit Optimierungsvariablen, die Elemente des Euklidischen Raumes sind. Die Abbildung zwischen Optimierungsvariable und ihrer Lösung ändert sich nicht mit der Zeit.
- *Dynamisches Optimierungsproblem:* Minimierung oder Maximierung einer Funktion mit Optimierungsvariablen, die Elemente des Hilbert-Raumes sind (z. B. Zeitfunktionen). Die Abbildung zwischen Optimierungsvariable und

ihrer Lösung ändert sich mit der Zeit. Hier wird nicht garantiert, dass die gleiche Optimierungsvariable in verschiedenen Zeitstempeln die gleiche Lösung liefert.

Definition 2.1.2 (Fitnesslandschaft) *Fitnesslandschaft wurde als Begriff in der Evolutionsbiologie durch Sewall Wright im Jahr 1932 eingeführt [155] und definiert einen der wichtigsten Aspekte in der Optimierung. Allgemein wird die Fitnesslandschaft auch Wertelandschaft genannt. Die Wertelandschaft ist die mathematische Funktion $f : M \rightarrow \mathbb{R}$, wobei M eine frei wählbare Menge ist. Der Wertelandschaft wird in der Optimierung eine Nachbarschaftsfunktion zugeordnet (siehe Abs. 2.1.1). Die Abbildung 2.1 gibt zwei Beispiele, wie eine Fitnesslandschaft aussehen könnte.*



(a) Eine 1-dimensionale Fitnesslandschaft **(b)** Eine 2-dimensionale Fitnesslandschaft

Abbildung 2.1 – Fitnesslandschaften zwei verschiedener Funktionen. Die Variable x definiert den Suchraum einer Funktion. $f(x)$ definiert den dazugehörigen Wert der Zielfunktion.

Das *allgemeine Minimierungsproblem* (Gl. 2.1) und das *allgemeine Maximierungsproblem* (Gl. 2.2) sind selbst wie folgt definiert:

$$min\ f(x), x \in S \tag{2.1}$$

$$max\ f(x), x \in S \tag{2.2}$$

wobei S eine Menge, der sogenannte *Suchraum*, ist. Die Zielfunktion f ist eine Abbildung $f : S \rightarrow \mathbb{R}$. Der Suchraum S definiert die Teilgebiete der Optimierung wie folgt [80]:

- S endlich: Kombinatorische bzw. Integer Optimierung
- $S \subseteq \mathbb{Z}^D$: Ganzzahlige Optimierung
- $S \subseteq \mathbb{R}^D$: Nichtlineare Optimierung
- S endlich und $S \subseteq \mathbb{R}^D$: Mixed-Integer Optimierung

wobei $\boldsymbol{D} \in \mathbb{N}$ die *Dimension* des Problems gibt. Außerdem bestimmt die Art der Zielfunktion $\boldsymbol{f}$, ob es sich um eine *einkriterielle* oder *mehrkriterielle* Optimierung handelt. Es wird zwischen *online* und *offline* Optimierung unterschieden. Im Folgenden werden die Begriffe näher vorgestellt.

2.1.1 Die Art des Suchraums

Ein Optimierungsalgorithmus versucht, die bestmögliche Lösung zu finden. Diese Lösung muss sich in dem Suchraum S befinden. Der Suchraum selbst ist entweder über kontinuierliche (z. B. die Sphere Funktion [7]) oder diskrete Werte (z. B. das Traveling Salesman Problem [4]) definiert. Die diskreten Werte können außerdem endlich sein.

In beiden Fällen wird eine angemessene Distanzfunktion gebraucht, welche den Abstand zweier Punkte im Suchraum definiert. Diese Verhältnisse wiederum definieren die Form (sogenannte Fitnesslandschaft[11]) des Optimierungsproblems. Um die Distanz zweier Werte in einem kontinuierlichen Raum zu bestimmen, kann die euklidische Distanz benutzt werden. Sind die Punkte a und b durch die Koordinaten $a = (a_1, ..., a_D)$ und $b = (b_1, ..., b_D)$ gegeben, so ist die euklidische Distanz wie folgt definiert [128]:

$$\sqrt{\sum_{i=1}^{D}(a_i - b_i)^2} \tag{2.3}$$

Während die Bestimmung der Distanz in einem kontinuierlichen Raum relativ einfach ist, kann man das über den diskreten Raum nicht sagen. Verschiedene Operatoren können zu verschiedenen Lösungen führen. Eine Möglichkeit, den Abstand zweier Punkte zu bestimmen, wäre der Hilbert-Raum [151].

2.1.2 Die Art der Zielfunktion

Ein anderer wichtiger Aspekt in der Optimierung ist die Zielfunktion, welche die zu erreichenden Ziele definiert. Falls nur eine Zielfunktion vorhanden ist, handelt es sich um eine *einkriterielle* Optimierung. Ein Beispiel wäre, wenn man versucht die Stromkosten zu senken oder wenn eine Firma versucht den Umsatz eines Jahres zu maximieren. In der *einkriteriellen* Optimierung haben die Fitnesslandschaften meistens mehrere lokale Minima bzw. Maxima. Das führt dazu, dass das Problem des Stillstandes (siehe Abs. 1.2.2) gelöst werden muss.

Bei der *mehrkriteriellen* Optimierung werden mehrere Ziele (d. h mindestens zwei) verfolgt. Die Ziele können voneinander abhängen, sich widersprechen oder sogar voneinander unabhängig sein. Daraus folgt, dass die Lösung eines solchen Problems kein einzelner Punkt ist, sondern eine Menge an möglichen Lösungen, welche *Paretofront* genannt wird. Da die Ziele in Konflikt miteinander stehen können, ist es oft unmöglich, sie gleichzeitig zu erfüllen. Die mathematischen Relationen werden auch durch den Term *Dominanz* beschrieben. Sie ist wie folgt definiert [46]:

Ein Punkt $\boldsymbol{x} \in \boldsymbol{S}$ dominiert einen anderen Punkt $\boldsymbol{y} \in \boldsymbol{S}$, falls folgendes gilt:

$$\forall\ m \in \{1, 2, ..., D\}: \quad f_m(x) \leq f_m(y)\ und \tag{2.4}$$

$$\exists\ n \in \{1, 2, ..., D\}: \quad f_n(x) < f_n(y) \tag{2.5}$$

wobei $\boldsymbol{D}$ die Dimension (siehe Abs. 2.5) ist. Betrachtet man hier nur zwei Lösungen $\boldsymbol{x}$ und $\boldsymbol{y}$, dann gibt es drei mögliche Beziehungen:

1. $\boldsymbol{x}$ dominiert $\boldsymbol{y}$
2. $\boldsymbol{y}$ dominiert $\boldsymbol{x}$
3. $\boldsymbol{x}$ und $\boldsymbol{y}$ dominieren sich nicht

Die Lösung solcher Probleme ist die Approximation der *Paretofront*.
Die Abbildung 2.2 zeigt eine mögliche Lösung einer mehrkriteriellen Optimierung samt Paretofront (rote Linie). In dem oberen Bereich der Paretofront liegen die akzeptierbaren Lösungen und in dem unteren Teil die nicht akzeptierbaren Lösungen. Gute Lösungen liegen auf der Linie der Paretofront.

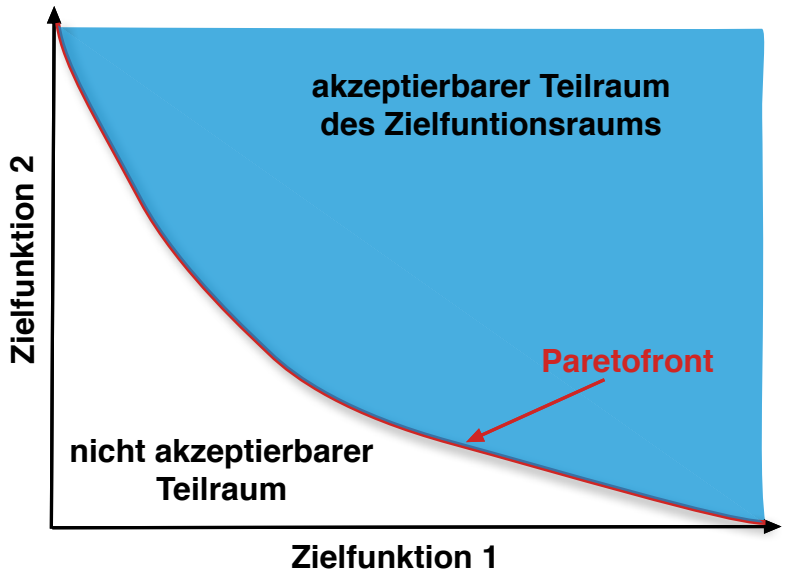


Abbildung 2.2 – Paretofront einer mehrkriteriellen Optimierung

2.1.3 Die Art der Optimierung anhand der Optimierungszeit

In der Optimierung werden meistens zwei Kriterien berücksichtigt, wenn es um die Effizienz der Lösung eines Algorithmus geht: Die Qualität der gefundenen Lösung und die Zeit, welche benötigt wurde, um diese Lösung zu finden. Das zweite Kriterium kann benutzt werden, um die Art der Optimierung zu klassifizieren. Es wird zwischen *online* und *offline* Optimierung unterschieden. Bei der ersten findet die Optimierung während der Laufzeit und bei der zweiten findet die Optimierung in der Entwurfszeit statt.

Diese erste Optimierung findet wie eine Schleife statt. In diesem Fall muss die Entscheidung über den nächsten Schritt in einer kurzen Zeit erfolgen. Die Abbildung 2.3 gibt einen Überblick dieses Prozesses. Die Variable t_e gibt die benötigte Zeit für die Evaluation eines Parameters aus dem Suchraum und die Variable t_o gibt die benötigte Zeit für die Optimierung an. Es findet immer ein Wechsel statt, der wie folgt läuft: Erst wird eine Testlösung zur Evaluation geschickt. Nachdem diese evaluiert wurde, wird der Wert der Zielfunktion zurückgeliefert. Der gelieferte Wert hat Einfluss auf das Auswählen der nächsten Testlösung. Das geht so weiter, bis eines der Abbruchkriterien erreicht wird. Die Abbruchkriterien werden vom Auftraggeber

angegeben. Einige Beispiele wären: (1) die Lösung hat eine ausreichende Qualität, (2) die zur Verfügung stehende Zeit ist abgelaufen oder (3) eine bestimmte Anzahl an Iterationen wurde überschritten.

In der realen Welt ist t_e deutlich größer als t_o. Deshalb wird die Gesamtzeit einer Optimierung von der Evaluationszeit dominiert und die Optimierungszeit t_o kann vernachlässigt werden. D.h., dass eine große Anzahl an Evaluationsschritten zu einer großen Evaluationszeit führt. Fast alle Optimierungsalgorithmen (z. B. PSO) arbeiten nach diesem Prinzip.

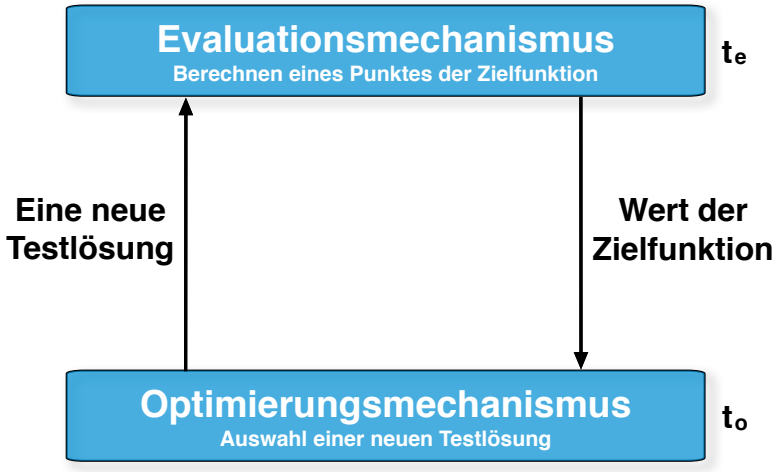


Abbildung 2.3 – Online Optimierung. Die abstrakte Darstellung eines Optimierungsprozesses aus [17].

Die zweite Art der Optimierung wird als offline Optimierung bezeichnet und findet nur am Anfang statt und wird in der Regel nur einmal durchgeführt. Hierbei existiert keine Interaktion zwischen dem Optimierungsmechanismus und der Umgebung. Die Eingangsvariablen müssen vor Beginn der Optimierung bekannt sein, etwas das in der realen Welt selten möglich ist. Die *offline* Optimierung wird z. B. sehr oft in der Domäne des Data Mining benutzt [144]. Durch Data-Mining wird zusätzliches Wissen anhand vorhandener Daten gewonnen. Da sowohl die genaueren Daten, der Einfluss der Parameter und ihre Anzahl, als auch das Ziel bekannt sind, ist eine *offline* Optimierung möglich.

2.2 Blackbox-Optimierung

In den oberen Abschnitten wurden die Optimierungsprobleme anhand ihrer Charakteristika (z. B. Art der Zielfunktion oder des Suchraums) klassifiziert. Es gibt aber vier große Optimierungsklassen, die über mathematische Eigenschaften der Zielfunktion klassifiziert werden. Die oben genannten Arten können sich in jeder dieser Klassen befinden:

1. **Second-order Optimierung:** Die Funktion $f(x)$, sowie die Steigung (1. Ableitung) und die Krümmung (2. Ableitung) sind gegeben.

2. **First-order Optimierung:** Nur die Funktion $f(x)$ und die Steigung sind bekannt.

3. **Derivative-free Optimierung:** Hier ist nur die Funktion $f(x)$ gegeben und wir wissen, dass diese stetig ist.

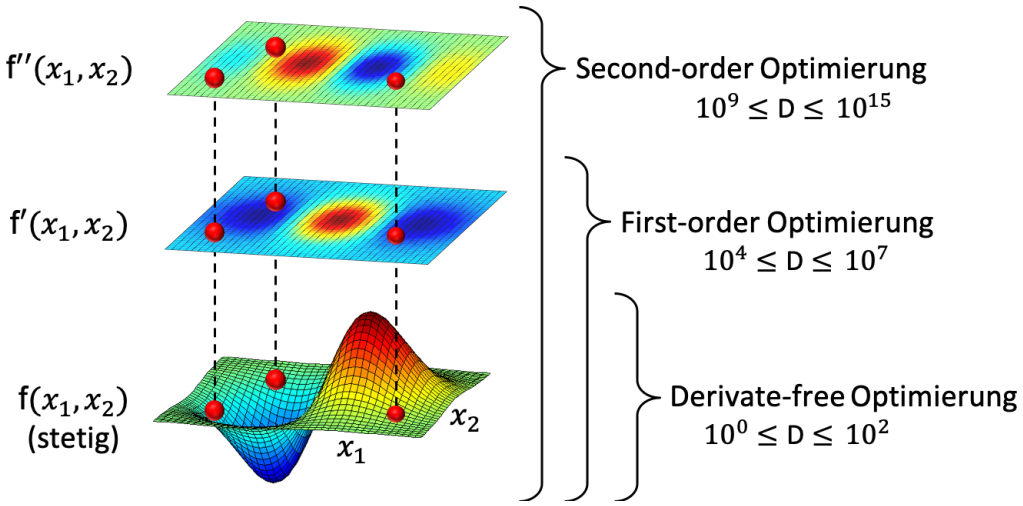


Abbildung 2.4 – Die drei mathematischen Arten der Optimierung.

4. **Blackbox-Optimierung:** Die Funktion$f(x)$ ist nicht bekannt. Zur Verfügung stehen der Werte- und Definitionsbereich und die Nebenbedingungen.

Die Probleme der ersten drei Klassen werden durch mathematische Optimierung (z. B. Trajektorienoptimierung) gelöst. Eine grafische Darstellung der ersten drei Klassen gibt die Abb. 2.4. Wie schon aus der Abbildung zu sehen ist, eignet sich die

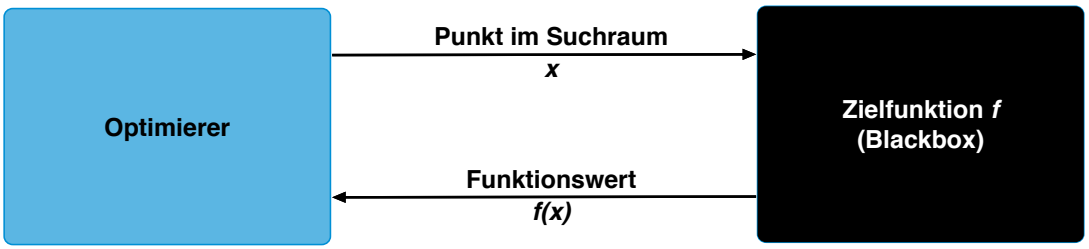


Abbildung 2.5 – Blackbox-Optimierung. Grafikdarstellung aus [130].

Derivative-free Optimierung für kleinere Probleme (Dimension zwischen 0 und 2) und die *Second-order Optimierung* für größere Probleme (Dimension zwischen 9 und 15). Mit anderen Worten: Je mehr Informationen wir über ein Problem haben, desto einfacher ist die Lösung dieses Problems.

Die vierte Klasse wird als Blackbox-Optimierung bezeichnet. Die Zielfunktion f ist nicht explizit angegeben und es stehen keine zusätzlichen Informationen über ihre Struktur zur Verfügung, z. B. die Ableitung oder Steigung. Die einzige Möglichkeit, solche Probleme zu optimieren, ist durch eine punktuelle Auswertung. Abb. 2.5 gibt einen Überblick über die Blackbox-Optimierung.

Als Blackbox-Probleme können auch Probleme gehandelt werden, für welche nur teilweise Informationen zur Verfügung stehen oder wenn das Fachwissen über sie fehlt.

2.3 Abgrenzung in der Optimierungswelt

In diesem Kapitel wurde ein grober Überblick über die Welt der Optimierung gegeben. Jeder dieser Bereiche bildet ein Gebiet in der Forschung ab. Die Abbildung 2.6 gibt einen Überblick der vorgestellten Optimierungsprobleme und deren Beziehungen [150]. Die vorgestellten Methoden und Werkzeuge in dieser Dissertation begrenzen sich nur auf folgende Optimierungsklassen und -probleme:

Statische Optimierungsprobleme: In dieser Arbeit werden nur die statischen Optimierungsprobleme, deren Fitnesslandschaft sich mit der Zeit nicht ändert, behandelt. Die vorgestellten Methoden und Werkzeuge gelten nur für diese Art der Probleme.

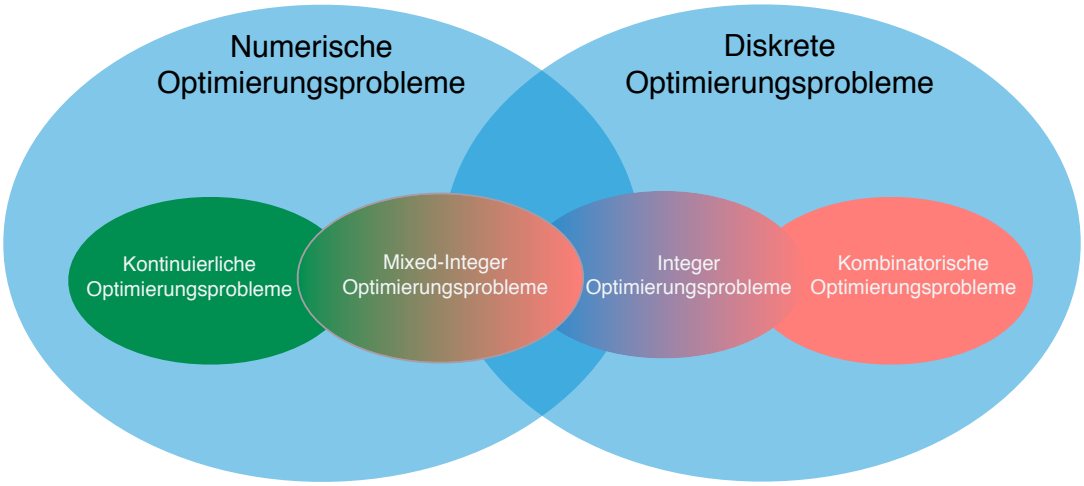


Abbildung 2.6 – Eine grobe Skizze der Problemklassen und deren Beziehungen. Forschungsfelder dieser Arbeit sind die kontinuierlichen Optimierungsprobleme als Teilmenge der numerischen Optimierungsprobleme.

Kontinuierliche, reellwertige Optimierung: Es werden nur Probleme, welche einen kontinuierlichen und reellwertigen Wertebereich haben, behandelt und evaluiert. Ob der Suchraum nichtlinear oder linear ist, ist unwichtig. Beide Fälle werden abgedeckt. Die vorgestellten Methoden funktionieren sehr gut auch für Integer (kombinatorisch) und Mixed-Integer Optimierung. Es ist nur eine Anpassung in dem Framework PASS nötig, um die gelieferten Werkzeuge auch passend zu machen.

Einkriterielle Optimierungsprobleme: Die behandelten Probleme haben nur ein einzelnes Optimierungsziel. Die mehrkriterielle Optimierung bildet ebenfalls eine sehr große Klasse von Optimierungsproblemen, die genauso erforscht wird. Es gibt aber Methoden, welche die Umwandlung von mehrkriteriellen Optimierungsproblemen in einkriterielle Aufgaben ermöglichen. Zwei davon sind die *gewichtete Summenmethode* [46] und die ϵ*-Constraint-Methode* [103]. Durch diese Methoden können mehrkriterielle Optimierungsprobleme in einkriterielle umgewandelt und durch PASS gelöst werden.

Offline/Online Optimierung: Es wird sowohl *online* als auch *offline* Optimierung betrieben. Der Algorithmus, welcher den Kern von PASS bildet, optimiert *online* und die *offline* Optimierung wird benutzt, um die Konfiguration der

Soft- und Hardware zu definieren. Eine genauere Erläuterung folgt im Kapitel 5.

Blackbox-Probleme: Die für diese Arbeit relevante Problemklasse ist die Klasse der Blackbox-Probleme. Diese Klasse ist heutzutage sehr relevant, da ihr großteils reale Problem angehören. Durch die steigende Komplexität ist es immer schwieriger, einen guten Überblick zu haben, deshalb werden die Probleme der realen Welt – auch wenn wir teilweise Informationen besitzen – der Klasse der Blackbox-Probleme zugeordnet.

2.4 Eigenschaften der behandelten Probleme

Nachdem genau definiert wurde, welche Optimierungsprobleme in dieser Arbeit behandelt werden, folgt hier ein Überblick über ihre Charakteristika [77].

Hat ein Optimierungsproblem nur ein Optimum, so spricht man von einem *unimodalen* Optimierungsproblem. Gibt es mehr als nur eins, dann spricht man von einem *multimodalen* Optimierungsproblem. Sind die Optima gleichmäßig verteilt, so sagt man, die Funktion habe eine *starke Struktur*. Somit impliziert der Begriff der starken Struktur auch, dass es sich um ein *multimodales* Problem handelt [40]. Außerdem ist an dieser Stelle noch der Begriff der *Separierbarkeit* einzuführen. Wird ein mehrdimensionales Problem betrachtet und lässt sich dieses in mehrere 1-dimensionale Probleme aufteilen, so heißt dieses Problem *separierbar*. Hat die minimale Änderung einer der Eingaben erhebliche Auswirkungen auf den Wert der Funktion, so sagt man, das Problem ist schlecht *konditioniert*.

2.5 Zusammenfassung

Wie man sehen kann, ist die Optimierung in vielen Gebieten vertreten. Jedes Gebiet benutzt seine eigenen Fachwörter für die gleiche Sache. Um Unregelmäßigkeiten zu vermeiden, werden hier die Notationen aus dem Gebiet der Optimierung definiert, welche im Laufe der Arbeit benutzt werden. Die wichtigsten Begriffe werden definiert und ihre genauere Bedeutung erläutert.

Optimierung: Da sich jedes Maximierungsproblem mit Zielfunktion f in ein Minimierungsproblem mit Zielfunktion $-f$ umformen lässt, werden im Folgenden

nur **Minimierungsprobleme** betrachtet und mit *Optimierung* ist immer das Minimieren gemeint. Wenn es sich um ein Maximierungsproblem handelt, wird das explizit erwähnt.

Dimension $D \in \mathbb{N}$: Mit Dimension eines Problems ist die Anzahl der Eingangs- oder Ausgangsvariablen gemeint. Je höher die Dimension ist, desto schwieriger ist die Suche nach dem Optimum, da der Abstand zwischen zwei Punkten mit höheren Dimensionen exponentiell steigt.

Zielfunktion f: Den Wert der Zielfunktion werden wir *Fitnesswert* $f \in \mathbb{R}$ nennen. In der einkriteriellen Optimierung hat dieser Wert nur eine Dimension.

***„Good enough“* Wert:** Bei den Blackbox-Problemen weiß man in der Regel nicht, wo das Optimum liegt. Das (gewünschte) optimale Minimum wird meistens von dem Anwender angegeben. Dieser Wert dient als Referenz und wird als *good enough* Wert bezeichnet und als Lösung akzeptiert. Es kann in der Regel nicht gesagt werden, ob dieser Wert überhaupt erreichbar ist oder ob dieser Wert das globale Minimum ist.

Suchraum S: Der Suchraum wird auch Zulässigkeitsbereich gennant. Jedes $x \in S$ ist eine *akzeptierte* Lösung. Der Suchraum wird wie folgt definiert:

$$S := \{x \subseteq \mathbb{R}^D | a \leq x \leq b\} \tag{2.6}$$

wobei $a \in \mathbb{R}^D$ und $b \in \mathbb{R}^D$ eine Intervallbeschränkung des Suchraums als Nebenbedingung angeben.

Def. Optimierungsproblem: Im Abschnitt 2.1 wurde das allgemeine Minimierungsproblem definiert. Wie im Abschnitt 2.3 angegeben, beschäftigt sich diese Arbeit nur mit *kontinuierlichen*, *reellwertigen* und *statischen* Problemen. Daher wird das Optimierungsproblem für diese Problemklasse wie folgt definiert [65]:

$$min\ f(x) | x \in S \tag{2.7}$$

wobei $x \subseteq \mathbb{R}^D$ und alle Funktionen vom $\mathbb{R}^D$ nach $\mathbb{R}$ gehen. Ist ein Optimierungsproblem ohne die Intervallbeschränkung gegeben, spricht man von

einem unbeschränkten Optimierungsproblem. Im allgemeinen Fall, d. h unter Berücksichtigung der Nebenbedingungen, handelt es sich um ein beschränktes Optimierungsproblem.

Die Validierung des Verfahrens mittels PASS erfolgt anhand von zwei Familien von Problemen. Teil der ersten Familie sind die synthetischen Benchmark-Probleme. Es wurden hier acht Probleme gewählt, mit denen alle Charakteristika (siehe Abs. 2.4) abgedeckt werden.

Die zweite Familie ist aus der realen Welt. Hier werden Probleme von Missionen im Weltall behandelt, bei denen die interplanetaren Bahnkurven bestimmt werden müssen. Diese Probleme sind von der European Space Agency (ESA)[1] und National Aeronautics and Space Administration (NASA)[2] veröffentlicht worden.

In dem nächsten Kapitel wird der Stand der Technik der Optimierungsalgorithmen vorgestellt.

[1] https://www.esa.int/ESA
[2] https://www.nasa.gov/

3

Stand der Technik

Auf dem Weg zum Optimum

Manchmal, wenn man Innovationen angeht, macht man Fehler. Es ist am besten, diese zuzugeben und damit weiterzumachen, andere Innovationen zu verbessern.

STEVE JOBS

In diesem Kapitel werden die Algorithmen behandelt, welche für diese Arbeit wichtig sind. Es gibt eine sehr große Anzahl an möglichen Algorithmen, deshalb ist eine Betrachtung aller Algorithmen nicht möglich. Die wichtigsten Repräsentanten jeder Klasse wurden ausgesucht, welche auch nach dem Stand der Technik die besten Ergebnisse liefern. Fast alle Algorithmen haben Gemeinsamkeiten und Unterschiede und anhand dieser wird auch die Klassifizierung durchgeführt. Zum Schluss wird gezeigt, wie die in Kapitel 1 vorgestellten Probleme von diesen Algorithmen behandelt werden und wo der Unterschied zwischen ihnen und PASS liegt.

3.1 Klassifizierung der Optimierungsalgorithmen

Im vorherigen Kapitel wurde genau definiert, welche Problemklassen Teil dieser Forschungsarbeit sind. Um numerische Optimierungsprobleme zu lösen, werden generell zwei verschiedene Klassen von Optimierungsverfahren angewendet. Die *deterministischen* Verfahren bilden die erste Klasse und die zweite Klasse wird von den *stochastischen* Verfahren gebildet. Sehr bekannte deterministische Verfahren sind: Das Gauß-Seidel-Verfahren [67] oder die sogenannten Gradientenverfahren [104]. Diese Algorithmen sind sehr gut geeignet, wenn es sich um kleine Probleme handelt. Dagegen sind die stochastischen Optimierungsverfahren sehr gut für Probleme größerer Dimensionen geeignet. Diese Algorithmen basieren auf verschiedenen Metaheuristiken (siehe Def. 3.1.1) und Zufallsfaktoren. Ein Unterschied zwischen den beiden Verfahren ist, dass die deterministischen Algorithmen bei gleicher Eingabe immer das gleiche Ergebnis liefern, was bei den stochastischen Methoden nicht der Fall ist.

Definition 3.1.1 (Metaheuristik) *Eine Metaheuristik ist ein Verfahren zur Lösung allgemeiner Optimierungsprobleme. Dieses Verfahren kombiniert Eigenschaften wie Zielfunktion und Heuristiken, um eine näherungsweise Lösung zu erreichen. Die Metaheuristik definiert eine abstrakte Folge von Schritten, die auf jedes Problem angewandt werden können [12].*

Es können Situationen entstehen, in welchen das Zufallsprinzip den Algorithmus daran hindert, das beste Optimum zu finden. Die Zufallstechnik hat aber auch Vorteile. Z. B. kann sie dazu führen, dass Suchräume untersucht werden, deren Untersuchung mit den deterministischen Verfahren nicht möglich wären.

Die behandelten Probleme dieser Arbeit sind aber durch eine sehr hohe Dimension und Komplexität charakterisiert, was dazu führt, dass sich nur auf die stochastischen Methoden konzentriert wird. Die stochastischen Verfahren wiederum unterteilen sich in zwei Klassen: Die *populationsbasierten* Algorithmen und die *trajektorienbasierten* Algorithmen. Die populationsbasierten Methoden haben die Natur als Vorbild. Wie der Name schon sagt, haben solche Methoden eine *Population* (siehe Def. 3.3.2), welche die Suche nach dem Optimum koordiniert. Diese Suche kann zum Erfolg führen, wenn die Individuen (auch *Partikel* oder *Agenten* genannt – siehe Def. 3.3.1) miteinander kooperieren und kommunizieren. Die trajektorienbasierten Algorithmen bedienen sich nur der Form der Fitnesslandschaft. Die

Abbildung 3.1 gibt einen kleineren Überblick dieser Klassen. Es gibt noch weitere Algorithmen, diese Arbeit beschränkt sich jedoch nur auf die Bekanntesten. Eine ausführliche Liste kann unter [150] gefunden werden.

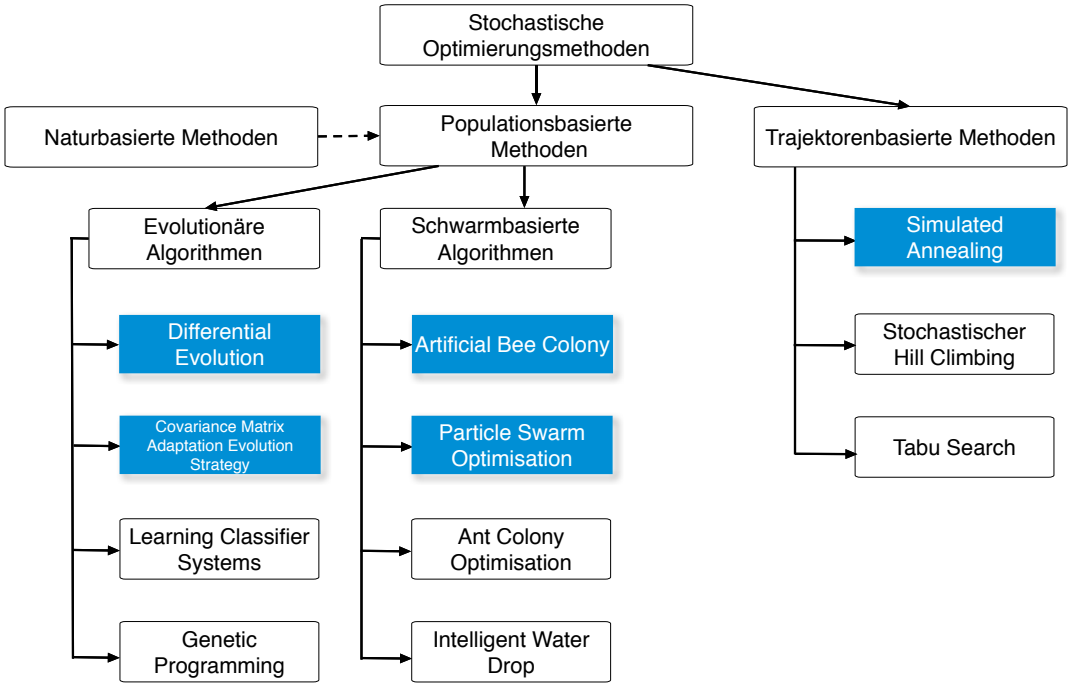


Abbildung 3.1 – Die Klassifizierung der stochastischen Optimierungsmethoden. Nur die blau markierten Algorithmen werden in dieser Arbeit näher untersucht und vorgestellt.

Die populationsbasierten Methoden unterteilen sich in zwei Algorithmengruppen. Die *evolutionären* Algorithmen und die *schwarmbasierten* Algorithmen. Repräsentanten der evolutionären Algorithmen sind: Differential Evolution (DE) [139], Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [69], Learning Classifier Systems (LCS) [96] und Genetic Programming (GP) [93]. Repräsentanten der schwarmbasierten Algorithmen sind: ABC [83], PSO [43], Ant Colony Optimization (ACO) [41] und Intelligent Water Drop (IWD) [132]. In der Klasse der trajektorienbasierten Methoden können folgende Algorithmen genannt werden: Simulated Annealing (SA) [29], Stochastic Hill Climbing (SHC) [110] und Tabu Search (TS) [31].

3.2 Trajektorienbasierte Methoden

```
    Eingabe f: Die Zielfunktion, welche minimiert werden muss
    Eingabe terminationCriterion: Das Abbruchkriterium
    Variable p_new: Der neue Lösungskandidat
    Variable p*: Der bist jetzt beste gefundene Lösungskandidat
    Ausgabe x: Die beste Lösung

1   BEGIN
2   p*.x ⟵ init()
3   while ¬terminationCriterion() do
4     p_new.x ⟵ newSample(p*.x)
5     if f(p_new.x) ≤ f(p*.x) then p* ⟵ p_new
6   end
7   return p*.x
```

Pseudocode 3.1 – Hill Climbing

Trajektorienbasierte Algorithmen starten mit einem zufälligen Lösungskandidaten und diese Lösung wird Schritt für Schritt durch eine andere, meistens bessere Lösung, ersetzt. Die Suche nach der nächsten Lösung liegt in der Regel in den benachbarten Gebieten im Suchraum, deshalb sind solche Algorithmen sehr gut für eine lokale Suche geeignet. Einer der prominentesten Repräsentanten dieser Klasse ist der Hill Climbing (HC) Algortihmus [126]. Wie der Name schon sagt (auf dt. Bergsteiger), besteht eine Analogie zu einem Bergsteiger, der versucht den Gipfel eines Berges zu erreichen. Die Sicht ist durch einen dichten Nebel versperrt (Analogie zu Blackbox) und die Idee ist, immer bergauf zu gehen. Er macht Schritt für Schritt weiter, bis es in alle Richtungen nur noch nach unten geht. Der HC Algorithmus ist deterministisch und benutzt keine zufälligen Elemente. Allerdings dient der HC Algorithmus als Basis für weitere stochastische Algorithmen wie der SHC [110] und SA.

Um die Idee der trajektorienbasierten Methoden nachzuvollziehen, wird ein Blick auf den HC geworfen. Der Pseudocode 3.1 [150] zeigt, wie HC arbeitet. Es wird mit einem zufälligen Lösungskandidat (Zeile 2) gestartet und nach jedem Schritt wird die Nachbarschaft nach einer besseren Lösung durchsucht. Wird diese gefunden, dann wird die alte Lösung ersetzt (Zeile 7). Der HC Algorithmus besitzt nur einen einzigen Agenten.

3.2.1 Simulated Annealing

SA lässt sich als *simulierte Abkühlung* übersetzen und ist ein heuristisches Approximationsverfahren, ähnlich wie der HC. Der Algorithmus wurde im Jahr 1983 durch S. Kirkpatrick, C. D. Gelatt und M. P. Vecchi vorgestellt [90]. Als Grundidee für SA dient die Nachbildung eines Abkühlungsprozesses. Nach dem Erhitzen eines Metalls sorgt die langsame Abkühlung dafür, dass die Atome ausreichend Zeit haben, sich zu ordnen und stabile Kristalle zu bilden. Durch diesen Prozess kann ein ideales Optimum erreicht werden. Die Temperatur entspricht der Wahrscheinlichkeit, mit der ein Zwischenergebnis auch schlechter werden darf. Im Vergleich zum HC kann dieser Algorithmus aber ein lokales Minimum wieder verlassen. Mit anderen Worten wird – wie in vielen stochastischen Verfahren – eine schlechtere Lösung zwischenzeitlich mit der Hoffnung akzeptiert, eine bessere zu finden.

Der Pseudocode 3.2 erläutert den SA Algorithmus [150]. Während der Optimierung nimmt die Temperatur T ab. Wenn die Temperatur am Anfang höher war, wird eine schlechtere Lösung oft angenommen. Hingegen werden bessere Lösungen immer akzeptiert. So ähnelt der Algorithmus einem *Random Walk* [119]. Wenn die Temperatur weiter abnimmt, ist die Chance, dass schlechtere Lösungen akzeptiert werden, kleiner. Wenn sie gleich Null ist, dann wird keine schlechtere Lösung akzeptiert, ähnlich wie der HC Algorithmus. Mit anderen Worten nutzt SA die Vorteile beider Verfahren: *Random Walk* und HC. Auf der einen Seite wird sehr viel in der Breite gesucht, um interessante Lösungen zu finden (Ähnlichkeit mit *Random Walk*), auf der anderen Seite besteht die Hoffnung, dass sich der Algorithmus während der Suche in einem bestimmten Bereich konzentriert und dort die beste Lösung findet (Ähnlichkeit mit HC).

```
Eingabe f: Die Zielfunktion, welche minimiert werden muss
Variable p_new: Der neue Lösungskandidat
Variable p_curr: Der Punkt, welcher momentan in dem Suchraum untersucht wird
Variable T: Die Temperatur des Systems, welche mit der Zeit abnimmt
Variable t: Der aktuelle Zeitpunkt
Variable ΔE: Die Energiedifferenz zwischen p_curr.x und p_new.x
Ausgabe x: Die beste Lösung

1  BEGIN
2    p_curr.x ⟵ init()
3    x ⟵ p_curr.x
4    t ⟵ 1
5    while ¬terminationCriterion() do
6        T ⟵ getTemperature(t)
7        p_new.x ⟵ newSample(p_curr.x)
8        ΔE ⟵ f(p_new.x − p_curr.x)
9        if ΔE ≤ 0 then
10           p_curr ⟵ p_new
11           if f(p_curr.x) < f(x) then x ⟵ p_curr.x
12       else
13           if randomUni[0,1) < e^(−ΔE/T) then p_curr ⟵ p_new
14   t ⟵ t + 1
15   end
16 return p*.x
```

Pseudocode 3.2 – Simulated Annealing

SA wird sowohl für die Lösung von diskreten als auch von kontinuierlichen Problemen verwendet. Die Anwendungsgebiete reichen vom Gebiet der Datenbanken [38] über Logistik [32] und Computergrafik [58, 143] bis hin zum Militär [75] .

3.3 Populationsbasierte Methoden

Die populationsbasierten Algorithmen arbeiten mit einer Population (Def. 3.3.2), welche selber aus mehreren Individuen, sogenannten Agenten (Def. 3.3.1) besteht. Die Agenten interagieren miteinander und tauschen Informationen aus, welche dazu dienen können, eine bessere Lösung zu finden. Wie genau der Austausch stattfindet, hängt von dem jeweiligen Algorithmus ab. Dort wird immer genau definiert, welche Agenten mit welchen kommunizieren dürfen, welche Information genau getauscht wird und wie oft dieser Tausch stattfindet. Dadurch besitzen die Agenten zwei Arten von Informationen: Ihre eigene und die der anderen Agenten. Die Informationen

werden dann benutzt, um bei jeder Iteration der Lösung näher zu kommen. Die Güte der Lösung haben wir im Kapitel 2 als *Fitnesswert* definiert.

Definition 3.3.1 (Agent *A*) *Der Agent ist das atomare Teil innerhalb eines populationsbasierten Algorithmus. Der Agent kann auch andere Bezeichnungen haben, beim PSO heißt er z. B. Partikel.*

Definition 3.3.2 (Population *Pop*) *Die Population ist definiert als Menge aller Agenten.* $Pop = \{A_o, ..., A_{n-1}\}$. *Die Agenten einer Population können nach Bedarf untereinander kommunizieren und Informationen austauschen.*

Die Grundidee ist, dass am Anfang des Verfahrens eine sehr breite Suche über den ganzen Suchraum erfolgt, um möglichst interessante Gebiete zu entdecken. Zu späteren Zeitpunkten werden diese Gebiete genauer Untersucht. Dafür werden die Begriffe *Exploitation* (Def. 3.3.3) und *Exploration* (Def. 3.3.4) verwendet. Ein Problem in dieser Algorithmenklasse ist das Exploration-Exploitation Dilemma [101], das darin besteht, die beste Aufteilung von Explorationsphasen und Exploitationsphasen zu finden. Jedes Verfahren benutzt verschiedene Methoden, um dieses Problem möglichst effizient zu lösen.

Definition 3.3.3 (Exploitation) *Die Exploitation hat als Ziel, vielversprechende Regionen des Suchraums näher zu untersuchen, um eine noch bessere Lösung als die aktuelle zu finden. Hier wird eine lokale Suche in den benachbarten Gebieten durchgeführt.*

Definition 3.3.4 (Exploration) *Die Idee von Exploration besteht darin, einen viel größeren Teil des Suchraums mit der Hoffnung zu erforschen, andere vielversprechende Lösungen zu finden. Wir haben es hier mit einer Diversifizierung der Suche zu tun, um nicht in einem lokalen Optimum stecken zu bleiben. Hier wird eine globale Suche im Suchraum durchgeführt.*

Im Folgenden werden zwei Hauptklassen der populationsbasierten Algorithmen angeschaut und für jede Klasse werden jeweils zwei sehr bekannte Algorithmen erläutert und im Detail erklärt.

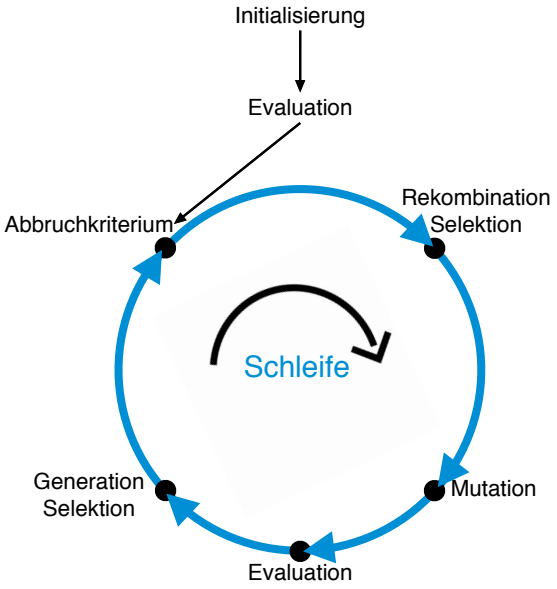


Abbildung 3.2 – Schematische Darstellung eines EA.

3.3.1 Evolutionäre Algorithmen

Die EA sind stochastische, metaheuristische Optimierungsalgorithmen, deren Funktion von der Evolution natürlicher Lebewesen motiviert ist. Sie gehören zu der Klasse der naturanalogen Verfahren. Die ersten Implementierungen wurden schon im Jahr 1950 veröffentlicht [120] und dienen als Grundlage der heutigen Algorithmen. Diese Algorithmen basieren auf Verfahren wie Selektion, Rekombination, Mutation usw. und unterscheiden sich anhand der Nutzung dieser Methoden. Wie alle anderen stochastischen Optimierungsverfahren, finden die EA ebenso nicht immer die beste Lösung. Ein grober Ablauf (s. Abb. 3.2) solcher Algorithmen sieht wie folgt aus:

1. **Initialisierung**: Eine Gruppe von Agenten wird zufällig generiert.

2. **Evaluation**: Jedem Agenten wird ein Fitnesswert zugeordnet.

3. Solange das Abbruchkriterium nicht erreicht ist, sollte folgendes getan werden:

   (a) **Rekombination (Def. 3.3.5)**: Die Agenten werden hier selektiert und ggf. miteinander kombiniert.

(b) **Mutation (Def. 3.3.6)**: Die nächste Generation von Agenten wird zufällig geändert.

(c) **Evaluation**: Jedem Agent wird ein Fitnesswert zugeordnet.

(d) **Selektion (Def. 3.3.7)**: Eine neue Generation fängt an.

Beim EA wird der Suchraum auch als *Genom* bezeichnet. Die Elemente des Genoms werden *Chromosome* genannt.

Definition 3.3.5 (Rekombination) *Die Rekombination (engl. Crossover) ist die Generierung einer neuen Lösung, in der Regel anhand zwei älterer Lösungen. Es gibt verschiedene Rekombinationsmöglichkeiten wie* ***One-Point-Crossover*** *oder* ***Two-Point-Crossover****. Genauere Details können [138] entnommen werden.*

Definition 3.3.6 (Mutation) *Wenn keine neuen Lösungen mittels Rekombination möglich sind, dann wird die Mutation verwendet. Die neuen Werte werden mit einer bestimmten Wahrscheinlichkeit generiert. Eine Mutation, z. B. einer binären Zahl, würde durch Bit-Flips an zufälligen Stellen erfolgen. Genauere Details über die Mutation können [138] entnommen werden.*

Definition 3.3.7 (Selektion) *Durch die Rekombination oder die Mutation entstehen neue Individuen. Das führt dazu, dass die Population ständig wächst. Das ist etwas, was man nicht will. Um das zu verhindern, wird die Selektion nach einem definierten Abstand benutzt. Für die Selektion gibt es wiederum verschiedene Methoden. Zwei sehr bekannte sind:* ***Roulette-Wheel-Selection*** *und* ***Tournament-Selection****. Genauere Details können [138] entnommen werden.*

Die EA werden für die Lösung sowohl von diskreten als auch von kontinuierlichen Problemen verwendet. Anwendungsgebiete von EAs sind sehr unterschiedlich, unter anderem Telekommunikation [30] und Data Mining [59].

3.3.1.1 Differential Evolution

DE ist ein populationsbasierter Optimierungsalgorithmus, welcher zuerst von Storn und Price im Jahr 1995 vorgestellt wurde [139]. DE wird für die Optimierung

kontinuierlicher Probleme benutzt. DE hat sich als eine sehr zuverlässige Optimierungsstrategie für die Probleme erwiesen, welche in reellwertigen Vektoren kodiert werden können. Der Ablauf des Algorithmus ist im Pseudocode 3.3 gegeben [16].

```
    Eingabe f: Die Zielfunktion, welche minimiert werden muss
    Eingabe CR: Die Wahrscheinlichkeit einer Rekombination
    Eingabe F: Der Gewichtungsfaktor
    Eingabe Pop: Die Population
    Eingabe D: Die Dimension des Problems
    Ausgabe S_best: Die beste Lösung
1
2   BEGIN
3   Pop ⟵ init(D)
4   S_best ⟵ getBestSolution(Pop)
5   while ¬terminationCriterion() do
6     for(P_i ∈ Pop)
7       S_i ⟵ newSample(P_i, Pop, D, F, CR)  //Pseudocode 3.4
8       if(f(S_i) ≤ f(P_i))
9         newPop ⟵ S_i
10      else
11        newPop ⟵ P_i
12      end
13    end
14    Pop ⟵ newPop
15    evaluate(Pop)
16    S_best ⟵ getBestSolution(Pop)
17  end
18  return S_best
```

Pseudocode 3.3 – Differential Evolution

Der Ablauf eines DE Algorithmus ist einem Schwarm-Algorithmus sehr ähnlich (vgl. Ab. 3.3.2). Der einzige Unterschied ist die Suche nach einer neuen Position (Zeile 7). Die Idee hier ist, dass die Generierung der neuen Stichproben durch einen gewichteten Differenzvektor zwischen zwei Agenten einer Population und den Parametervektoren eines Dritten erfolgt. Der Pseudocode 3.4 gibt den genaueren Ablauf dieser Methode [16] an.

```
    Eingabe P_i: Ein Agent der Population
    Eingabe Pop: Die Population
    Eingabe CR: Die Wahrscheinlichkeit einer Rekombination
    Eingabe F: Der Gewichtungsfaktor
    Ausgabe S: Der nächste Lösungskandidat

1   BEGIN
2   repeat
3     P_1 ⟵ RandomMember(Pop)
4   until P_1 ≠ P_0
5   repeat
6     P_2 ⟵ RandomMember(Pop)
7   until P_2 ≠ P_0 ∨ P_2 ≠ P_1
8   repeat
9     P_3 ⟵ RandomMember(Pop)
10  until P_3 ≠ P_0 ∨ P_3 ≠ P_1 ∨ P_3 ≠ P_2
11  Cutpoint ⟵ RandomPosition(Pop)
12  S ⟵ 0
13  for(P_i ∈ Pop)
14    if(i ≡ CutPoint ∨ Rand() < CR)
15      S_i ⟵ P_3i + F × (P_1i − P_2i)
16    else
17      S_i ⟵ P_i
18    end
19  end
20  return S
```

Pseudocode 3.4 – Differential Evolution - Die *newSample* Funktion

Die Wahl der Parameter *F*, *CR* und *Pop* hat einen großen Einfluss auf die Performance des Algorithmus. Anwendungsgebiete von DE sind z. B. Finanzen [61], Logistik [20] und Motorsteuerung [19].

3.3.1.2 Covariance Matrix Adaptation Evolution Strategy

CMA-ES ist einer der bekanntesten Algorithmen in der Klasse der EA, welcher durch N. Hansen, S. Müller und P. Koumoutsakos im Jahr 2003 vorgestellt wurde [68]. Wie der Name schon sagt, basiert der CMA-ES auf einer derandomisierten Evolutionsstrategie mit einer Kovarianzmatrixanpassung der gaußschen Normalverteilung. Das Prinzip der Anpassung basiert auf der Idee, die Wahrscheinlichkeit von erfolgreichen Schritten, welche in der Vergangenheit erfolgt sind, zu erhöhen. Um das zu erreichen, wird die Kovarianzmatrix so verändert, dass die Wahrscheinlichkeit des selektierten Schrittes der letzten Generation deutlich ansteigt.

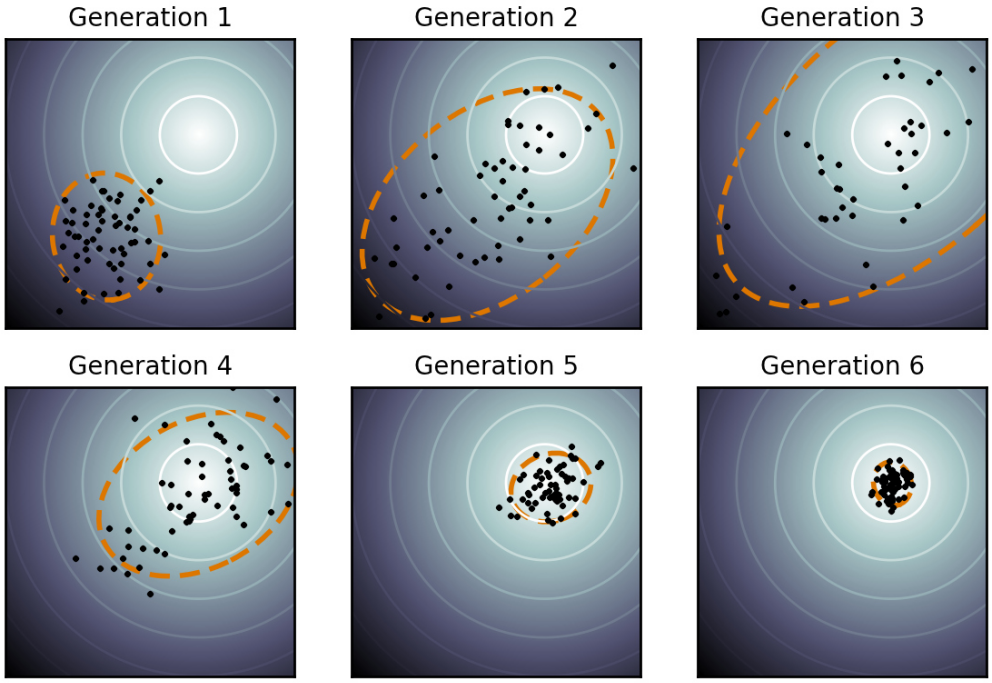


Abbildung 3.3 – Darstellung eines Optimierungsablaufs mit CMA-ES an einem einfachen zweidimensionalen Problem.

CMA-ES benutzt zwei Hauptprinzipien für die Anpassung der Parameter der Suchverteilung: Die Maximum-Likelihood-Methode [131], basierend auf der Idee, die Wahrscheinlichkeit erfolgreicher Kandidatenlösungen und Suchschritte zu erhöhen und Idee der zwei Pfade, welche die zeitlichen Entwicklungen der Strategie aufzeichnen. Diese Pfade enthalten wichtige Informationen über die verschiedenen Korrelationen von vorherigen aufeinanderfolgenden Schritten. Eine einfache Darstellung eines Optimierungsablaufs mit CMA-ES wird in der Abb. 3.3 gezeigt. Der Pseudocode 3.5 skizziert einen Ablauf dieses Algorithmus.

Anwendungsgebiete von CMA-ES sind z. B. Forensik [74], Vernetzung [78] und Engineering [57].

```
    Eingabe f: Die Zielfunktion, welche minimiert werden muss
    Eingabe λ: Anzahl der Stichproben per Iteration, min. 2, in der Regel > 4
    Eingabe m: Verteilungsmittelwert und der aktuelle Lösungskandidat
    Eingabe σ: Schrittgröße
    Eingabe C: Symmetrische Kovarianzmatrix
    Eingabe p_σ und p_c: Die zwei Entwicklungspfade (Isotropic und Anisotropic)
    Ausgabe m oder x_1: Die beste Lösung

1  set λ
2  initialise m, σ, C=I, p_σ=0, p_c=0
3  while ¬terminationCriterion() do
4      for(i ∈ {1....λ})
5          x_i = sample_multivariate_normal(mean=m, covariance_matrix=σ^2 C)
6          f_i = fitness(x_i)
7      x_{1...λ} ⟵ x_{s(1)...s(λ)} with s(i) = argsort(f_{1...λ}, i)  //Lösungen sortieren
8      m' = m
9      m ⟵ update_m(x_1,...,x_λ)  //bewege den Mittelwert zu einer besseren Lösung
10     p_σ ⟵ update_ps(p_σ, σ^{-1} C^{-1/2}(m − m'))
11     p_c ⟵ update_pc(p_c, σ^{-1}(m − m'), ||p_σ||)
12     C ⟵ update_C(C, p_c, (x_1 − m')/σ, ..., (x_λ − m')/σ)
13     σ ⟵ update_sigma(σ, ||p_σ||)
14 end
15 return m or x_1
```

Pseudocode 3.5 – Covariance Matrix Adaptation Strategy

3.3.2 Schwarmbasierte Algorithmen

Die schwarmbasierten Algorithmen werden durch eine dezentralisierte Struktur charakterisiert. Diese Struktur oder Population besteht aus mehreren Agenten, welche miteinander interagieren und verschiedene Informationen austauschen. Der Unterschied zu den anderen Klassen liegt darin, dass die Agenten nur eine eingeschränkte Sicht der Umgebung haben. Das führt dazu, dass ein einziger Agent aus dieser Population nicht *intelligent* genug ist, um das Problem alleine zu lösen. Hier ist die Zusammenarbeit mit den anderen Agenten erforderlich, um das Ziel zu erreichen. Wir haben es hier mit einem kollektiven Verhalten zu tun. Eine Charakteristik dieser Algorithmen ist, dass sie sich selbst organisieren, selbst adaptieren und sogar selbst neue Lösungen finden können [13]. Diese Methoden bestehen aus Schwärmen, deren Verhalten an die Natur angelehnt ist. In diesem Kapitel werden zwei typische Repräsentanten dieser Klasse vorgestellt: Der Artifical Bee Colony Algorithmus (ABC), welcher sich der Analogie von Bienen bedient, und der Particle

Swarm Optimization Algorithmus (PSO), welcher sich der Analogie von Vögel- oder Fischschwärmen bedient.

3.3.2.1 Artificial Bee Colony

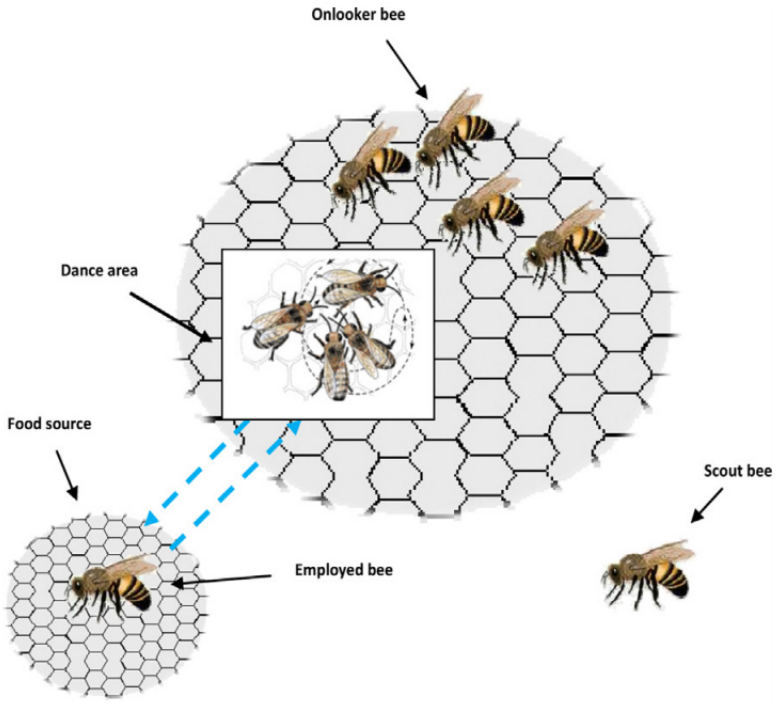


Abbildung 3.4 – Artificial Bee Colony. Grafische Darstellung der Elemente eines ABC Algorithmus aus [85].

ABC ist ein populationsbasierter Algorithmus, welcher im Jahr 2005 von Dervis Karaboga veröffentlicht wurde [83]. ABC wurde durch das intelligente Verhalten der Honigbiene motiviert. Ein weiterer Vorteil dieses Algorithmus ist die kleine Anzahl der Eingangsvariablen. Der Pseudocode 3.6 schildert die Funktionsweise des Algorithmus [107].

```
    Eingabe f: Die Zielfunktion, welche minimiert werden muss
    Eingabe X_i: Ein Agent mit lb_i ≤ X_i ≤ ub_i für i = 1,.....,D
    Eingabe SN: Populationsgröße
    Eingabe MCN: Maximale Anzahl der Zyklen
    Ausgabe: Die beste Lösung

    BEGIN
1   num_eval ⟵ 0
2   for s = 1 to SN do
3     X(s) ⟵ random solution
4     f_s ⟵ f(X(s))
5     num_eval++
6   end
7   cycle ⟵ 1
8
9   while cycle < MCN do
10    start employed_bee_phase();
11    start onlooker_bee_phase();
12    start scout_bee_phase();
13    Memorize the best solution achieved so far
14    cycle++
15  end
16 end
```

Pseudocode 3.6 – Artificial Bee Colony

Der ABC besteht aus drei Phasen. Die erste Phase employed bees weist Ähnlichkeiten zum PSO (siehe Abs. 3.3.2.2) auf: Jeder Agent bewegt sich auf genau einen zufälligen anderen Agenten zu. Anders als beim PSO bewegen sich ABC Agenten jedoch nur in einer einzigen Dimension; außerdem bleiben sie stehen, anstatt zu einer Position mit einem schlechteren Fitnesswert zu wechseln.

Die zweite Phase scout bees implementiert ein Neustart-Kriterium, das dann einsetzt, wenn sich ein Agent für mindestens durch die von *limit* vorgegebene Anzahl an Evaluationen nicht bewegt hat. Damit soll der Explorationsaspekt der Optimierung gewährleistet werden. Pro Iteration kann nur ein einziger Agent neu gestartet werden.

Die dritte Phase onlooker bees dient dazu, eine gute Exploitation zu erreichen, indem diejenigen Agenten die meisten Evaluationen erhalten, die global betrachtet an den besten bisher gefundenen Positionen stehen. Da in dieser Phase noch einmal *SN* Evaluationen durchgeführt werden, ist die Population *2 * SN* groß, wobei nur die Hälfte dieser Population mit einem Gedächtnis versehen ist. Die Abb. 3.4 gibt eine grafische Darstellung der Elemente des ABC Algorithmus.

Anwendungsgebiete von ABC sind z. B. Stromversorgungssysteme [71], Physik [136] und Software Testing [35].

3.3.2.2 Particle Swarm Optimization

Die PSO ist ein stochastisches populationsbasiertes Optimierungsverfahren, welches im Jahr 1995 von Kennedy und Eberhart vorgestellt wurde [44, 88]. Der Algorithmus basiert auf einem vereinfachten Modell des Verhaltens biologischer Organismen wie Vogel- oder Fischschwärmen. Die Zusammenarbeit der Organismen für das Auffinden einer bestimmten Position im Raum, z. B. einer Nahrungsquelle, wird mit der Suche nach einem Optimum in einem reellen Suchraum assoziiert. Bei den meisten Implementierungen der PSO bewegen sich die Agenten (hier Partikel genannt) im Suchraum, um eine Lösung zu finden. Die Richtung und Geschwindigkeit dieser Bewegung wird durch die eigene Information und die Information der anderen Mitglieder (Nachbarschaft - s. Def. 3.3.8) gesteuert. Die Berücksichtigung der eigenen Suchhistorie dient ebenfalls als zusätzliche Information. Aufgrund dieser beiden Informationen und der Information über die aktuelle Position ändert der Agent seine Bewegungsrichtung.

Definition 3.3.8 (Nachbarschaft) *Die Nachbarschaft ist definiert als die Menge aller Partikel, mit denen jeder Partikel in der Lage ist, zu kommunizieren. Diese Beziehung wird durch die Topologie definiert. Es gibt verschiedene Arten von Nachbarschaften, eine davon ist z. B. die globale Topologie, wo jeder Agent mit jedem anderen Agent kommunizieren darf.*

Ganz am Anfang des Algorithmus erfolgt eine Initialisierung der Population, die zufällig im Suchraum verteilt wird. Jeder Agent repräsentiert eine D-dimensionale Lösung im Suchraum und hat eine *Position*, eine *Geschwindigkeit* und eine definierte *Nachbarschaft*. Die Position und die Geschwindigkeit sind als Vektoren kodiert. Nach der Initialisierung berechnen die Agenten ihren Fitnesswert und aktualisieren die alten Positionen und Geschwindigkeiten. Das gleiche passiert jeweils mit dem globalen und persönlichen besten Fitnesswert. Nach der Aktualisierung wird geprüft, ob das Abbruchkriterium erreicht wurde. Falls ja, dann ist der Agent mit dem besten globalen Fitnesswert die gesuchte Lösung. Falls nicht, dann werden die Geschwindigkeiten und die Richtung der Partikel aktualisiert. Daraus folgt,

dass sich die Position ebenfalls ändert. Das wird dann iterativ wiederholt, bis das Abbruchkriterium erreicht wurde. Der Pseudocode 3.7 erläutert diesen Ablauf [150].

```
    Eingabe terminationCriterion: Das Abbruchkriterium
    Variable ps: Die Anzahl der Agenten in der Population
    Variable Pop: Die Population
    Variable i: Zähler
    Ausgabe x: Die beste gefundene Lösung

    BEGIN
1     pop ⟵ createPop(ps)
2     while ¬terminationCriterion() do
3       for i ⟵ 0 up to ps − 1 do
4         pop[i] ⟵ psoUpdate(pop[i], pop)
5       end
6     end
7   end
8   return best(pop).x
```

Pseudocode 3.7 – Particle Swarm Optimization

Die Einfachheit und Effizienz dieses Algorithmus ermöglicht ihm den Einsatz in vielen Gebieten wie: Data Mining [158], E-Learning [72], Finanzen [60] und Security [15].

3.4 Problemlösung

Im Kapitel 1 wurden die drei Hauptprobleme der Optimierungsverfahren aufgelistet. Hier werden diese Probleme genauer angeschaut und gezeigt, wie andere diese Probleme lösen, um dann den Unterschied zwischen den existierenden Verfahren und PASS zu zeigen.

3.4.1 Parallelisierung

Bevor mit dem Problem der Parallelisierung angefangen wird, muss ein genauerer Blick auf die parallelen Architekturen und deren Programmiermodelle geworfen werden. Dann wird das Problem der Parallelisierung untersucht und die existierenden Lösungen werden aufgelistet.

3.4.1.1 Parallele Architekturen und Programmiermodelle

Heutige Programmiermodelle lassen sich grob in drei Gruppen einteilen. Die oberste Klasse bildet die *globale Parallelisierung*. Hier haben wir es mit vielen sehr komplexen Rechnerknoten zu tun (in der Abb.1.1 als *Maschine X* bezeichnet) und die Kommunikation zwischen den Knoten findet mittels eines schnellen Verbindungsnetzwerks statt. Diese Knoten bilden zusammen mit dem Netzwerk einen so genannten High Performance Computing Cluster (HPCC). Zwei Beispiele für die Kommunikation hierfür sind die *Ethernet-Verbindung* [109] oder die *InfiniBand-Verbindung* [121]. Der Unterschied zwischen den Technologien liegt in der Geschwindigkeit der Übertragung und an der Latenz (s. Def. 3.4.1).

Definition 3.4.1 (Latenz) *Als Latenz wird die Zeit bezeichnet, die eine Nachricht für den Weg vom Sender bis zum Empfänger benötigt. Die Latenz beinhaltet drei Komponenten: (1) die Ausbreitungsverzögerung, (2) die Übertragungsverzögerung und (3) die Wartezeit [49].*

Die Tabelle 3.1 gibt einen Überblick über diese Unterschiede. Die Kommunikation zwischen den Knoten findet über Nachrichten statt. Die bekannteste und verbreitetste Methode, die *globale Parallelisierung* zu realisieren, ist die Nutzung der Kommunikationsbibliothek MPI (Message Passing Interface). Wenn wir eine Ebene tiefer gehen, dann haben wir es mit einer *grobkörnigen Parallelisierung* zu tun, welche innerhalb der Rechenknoten stattfindet. Heutige Rechner bestehen in der Regel aus mehreren Multi-Kern Central Processing Units (CPUs). Um die Parallelisierung auf dieser Ebene durchzuführen, hat sich die C/C++ Erweiterung OpenMP (Open Multi-Processing) als Standard durchgesetzt. Dabei wird das Programm auf den einzelnen Kernen der CPU ausgeführt. Die dritte Möglichkeit, um zu parallelisieren, wäre die *feinkörnige Parallelisierung* mittels GPGPUs. Eine GPGPU ist in mehrere Streaming Multi-Processors (SMs) unterteilt, die jeweils mehrere Kerne besitzen (s. Abb. 3.5). Die Kerne eines SM kommunizieren über einen lokalen gemeinsamen Speicher und die Kommunikation zwischen den SMs wird mittels eines globalen gemeinsamen Speichers, den sogenannten Dynamic Random Access Memory (DRAM), durchgeführt. Als Standard hat sich hier CUDA (Compute Unified Device Architecture) durchgesetzt.

Die letzte Ebene (*feinkörnige Parallelisierung*) wird in dieser Arbeit nicht verwendet und somit nicht genauer erklärt. Die beiden anderen Programmiermodelle werden detaillierter erklärt.

Tabelle 3.1 – Netzwerktechnologien in einem HPCC

Technologie	Bandbreite in $MByte/s$	Latenz in μ sec für 1 Byte
Hauptspeicher	>1000	<0,01
Fast Ethernet	11	70
GBit Ethernet	110	30
InfiniBand	805	7,5

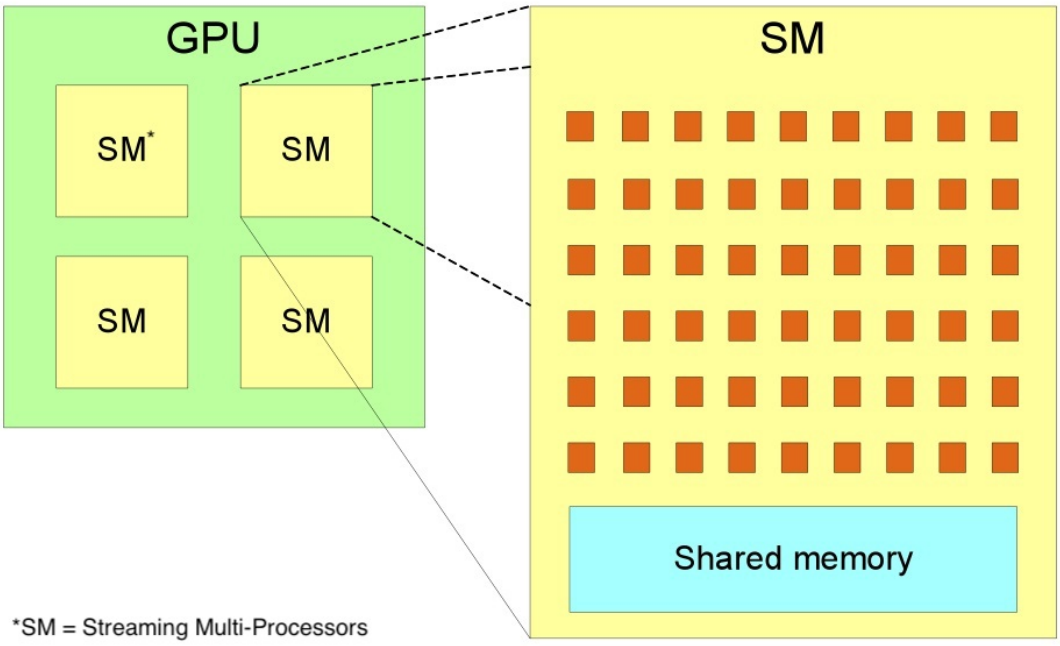


Abbildung 3.5 – GPGPU Architektur. Bild aus [55].

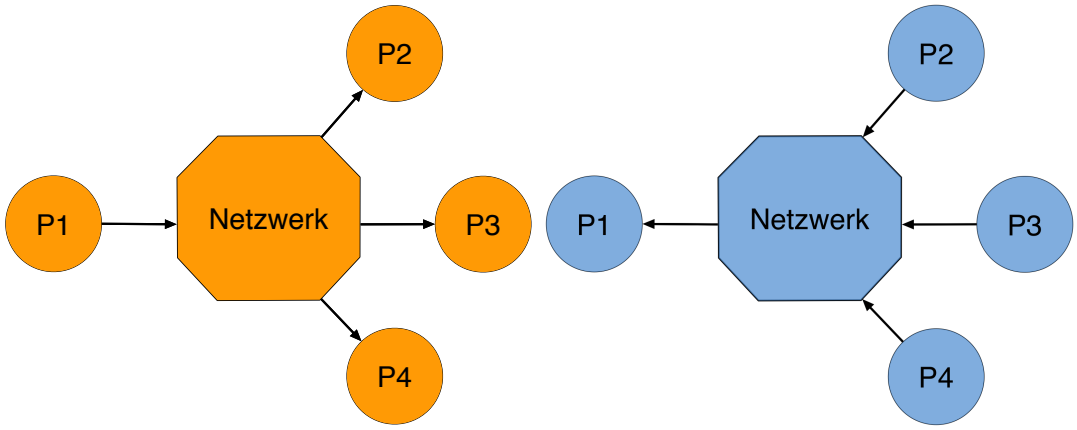
Das Grundproblem hier ist das Mapping. Auf der eine Seite haben wir den parallelen Rechner, welcher zur Verfügung steht, und auf der anderen Seite haben wir das Programm, welches parallelisiert werden soll. Als Ziel soll eine hohe Auslastung des Rechners mit der kleinstmöglichen Latenz erreicht werden. Mit anderen Worten, muss man versuchen, die Parallelisierung der Anwendung möglichst gut auf die parallele Architektur zu mappen.

MPI

Bei MPI handelt es sich um einen Standard, der den Nachrichtenaustausch bei parallelen Berechnungen auf verteilten parallelen Computern oder Clustern beschreibt. MPI selbst ist also keine Programmiersprache oder Implementierung, sondern eine Spezifikation von Ansammlungen von Operationen. Sie legt die Namen und Parameterlisten in den Sprachen C, C++ und Fortran fest [66]. Die ersten Arbeiten mit MPI fingen im Jahr 1991 an, aber die erste Veröffentlichung kam erst im Jahr 1994. Die Hauptidee von MPI ist die Lieferung einer Schnittstelle, welche sehr einfach zu benutzen ist.

So kann ein MPI-Block innerhalb eines Programms durch die MPI Operationen *MPI_Init* und *MPI_Finalize* initialisiert und abgeschlossen werden. Dieser Block wird beim Ausführen des Programms in einer vom Anwender festgelegten Anzahl von Prozessen parallel ausgeführt. Im MPI kommt die Kommunikation nur zu Stande, wenn Sender und Empfänger die entsprechende Operation durchführen. Wir haben drei Arten von Kommunikationen: Punkt-zu-Punkt, Punkt-zu-Mehrpunkt und Mehrpunkt-zu-Punkt.

In dem Fall von Punkt-zu-Punkt Kommunikation muss der sendende Prozess *MPI_Send* und der Empfänger *MPI_Recv* ausführen [66]. Somit ist die Kommunikation im MPI ein zweiseitiger Prozess. Punkt-zu-Mehrpunkt Kommunikation wird im MPI zum Beispiel durch die kollektive Operation *MPI_Broadcast* realisiert. Bei den kollektiven Operationen führen alle Prozesse, im Gegensatz zu den Punkt-zu-Punkt Operationen, dieselbe Operation aus. Dazu können Prozesse durch eine eindeutige Prozessnummer, dem *rank*, identifiziert werden. Infolgedessen kann der Empfänger bei einer Mehrpunkt-zu-Punkt-Operation durch seinen *rank* in der Parameterliste bestimmt werden. Die restlichen Prozesse werden somit zum Sender. Der empfangende Prozess erhält hierbei in einer Datenstruktur, oft einem Array, die Nachrichten aller anderen Prozesse. Die Abbildung 3.6 stellt diese Kommunikationsmodelle grafisch vor.



(a) Punkt-zu-Mehrpunkt Kommunikation **(b)** Mehrpunkt-zu-Punkt Kommunikation

Abbildung 3.6 – MPI Kommunkationsmodell. Die grafische Darstellung der zwei Modelle

OpenMP

OpenMP ist ein Programmiermodell zur speicherbasierten Parallelisierung auf Multiprozessor-Rechnern [34]. Die Kommunikation findet hier über einen gemeinsamen Speicher statt. OpenMP wurde erst im Jahr 1997 speziell für die Programmiersprache Fortran veröffentlicht. Im darauffolgenden Jahr wurde die Unterstützung für C und C++ bekannt gegeben. Heutzutage ist OpenMP Teil vieler Compiler und wird ständig mit neuen Funktionen erweitert. Die Standards werden durch eine Gemeinschaft geregelt, in denen große Firmen wie AMD, ARM, Fujitsu, HP, IBM, Intel, Oracle und Texas Instruments sitzen.

Die Motivation für die Einführung lag an der nicht trivialen Schwierigkeit bei der Programmierung und Synchronisierung von Prozessen in einer gemeinsamen Speicherarchitektur. Für das Markieren der zu parallelisierenden Abschnitte nutzt OpenMP verschiedene Direktiven. Die eingefügten Direktiven werden von den Compilern nicht beachtet, falls bei der Kompilierung nicht explizit OpenMP unterstützt wird. Somit ist keine Umschreibung des Programmcodes notwendig.

Ein OpenMP-Programm startet nur mit einem Thread, dem Master. Die Arbeiter-Threads werden zusammen mit dem Master-Thread in einer parallelen Region erzeugt. Hier wird die serielle Arbeit einzelner Threads parallel ausgeführt (Fork-Join-Model). Die Abbildung 3.7 gibt einem Überblick über diesen Prozess.

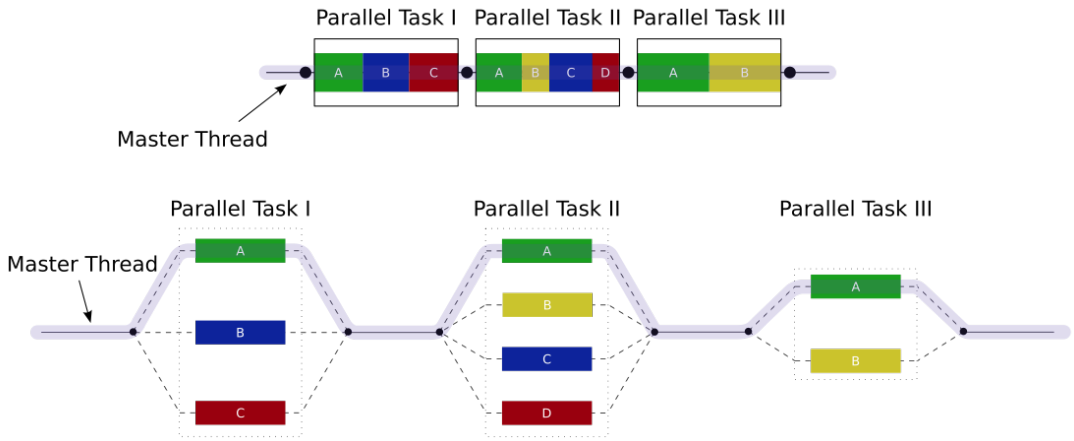


Abbildung 3.7 – Fork-Join Modell von OpenMP. Bild aus [54].

3.4.1.2 Stand der Technik: Parallelisierung

Die Struktur der populationsbasierten Algorithmen – bestehend aus einzelnen Agenten – und die beinahe Unabhängigkeit dieser liefern die optimalen Voraussetzungen für eine Parallelisierung. In diesen Algorithmen haben wir eine bestimmte Anzahl von Evaluationsschritten, welche durch eine von Anfang an festgelegte Anzahl von Agenten iterativ durchgeführt werden. Die Abbruchkriterien bestimmen die Dauer dieses Prozesses. Dabei führt jeder Agent die gleiche Operation auf jeweils verschiedenen Daten aus. Hier handelt es sich um das klassische Single Instruction Multiple Data (SIMD) Paradigma [52].

In der Literatur sind sehr viele verschiedene Ansätze zur Parallelisierung von populationsbasierten Algorithmen zu finden. Erick Cantu-Paz [18] hat die verschiedenen Ansätze zusammengefasst und teilt sie in drei verschiedene Kategorien ein: 1) *globale Parallelisierung*, 2) *grobkörnige Parallelisierung* und 3) *feinkörnige Parallelisierung* (vgl. Abs. 3.4.1.1). In der globalen Parallelisierung wird nur eine einzige Population verwendet und die Agenten dieser Population werden auf verschiedene Rechenknoten verteilt. Da der Overhead für die Kommunikation hier relativ groß ist, wird sie sehr beschränkt, sie findet nur am Anfang und Ende statt. Das Prinzip wird auch Master-Slave genannt. Der Master-Knoten ist für die Koordination verantwortlich und verteilt die Aufgaben und sorgt dafür, dass die Ergebnisse zum Schluss zusammengefasst werden. Um den Overhead (s. Def. 3.4.2) für die Kommunikation zu vermeiden, geht man eine Ebene tiefer und benutzt die grobkörnige Paralleli-

sierung. Hier werden sub-Populationen erzeugt und die Kommunikation ist durch den gemeinsamen Speicher deutlich schneller (s. Tab. 3.1). Die Parallelisierung wird hier entweder mittels MPI oder mittels OpenMP realisiert. Liu et. al [98] und O. Altinoz and A.E. Yilmaz [5] haben zwei Algorithmen, welche auf einer OpenMP Implementierung basieren, vorgestellt. Die feinkörnige Parallelisierung wird für hochparallele Rechner benutzt. Hier wird jeder Agent einem Kern zugeordnet, was dazu führt, dass wir einen sehr schnellen Kommunikationskanal brauchen, um den Overhead auf einem Minimum zu halten. Die meisten parallelen Implementierungen werden durch CUDA bewältigt. Einige Beispiele wären hier [79, 84]. Weiterhin gibt es zahlreiche Studien [95, 125], welche die verschiedenen Methoden einander gegenüber stellen und die Vor- und Nachteile auflisten.

Definition 3.4.2 (Overhead) *Overhead wird als eine Kombination aus unproduktiven Zeiten in der parallelen Programmierung definiert. Diese setzen sich zusammen aus Zeiten, welche die parallele Maschine nicht für das Ausführen des Algorithmus nutzt, sondern für andere Aufgaben wie z. B. Erzeugung von Threads, Kommunikationszeit, Wartezeit, Synchronisierung, Thread-Terminierung, usw..*

Eine ganz andere Art der Parallelisierung ist die automatisierte Parallelisierung. Hier gibt es verschiedene Frameworks wie: *PGI Accelerator* [118], *HMPP* [145] oder *R-Stream* [123], welche dem Benutzer helfen sollen, eine Parallelisierung durchzuführen, indem verschiedene Bereiche mit Direktiven markiert werden.

3.4.2 Stillstand

Die populationsbasierten Algorithmen sind durch ein zu frühes Stagnieren in lokalen Minima charakterisiert. Dieses Problem lässt sich nicht total vermeiden, deshalb wird versucht, das Problem des „Stillstands“ zu minimieren. Hierfür gibt es verschiedene Ansätze für die Realisierung. Sie basieren auf zwei Ideen: Der Idee den Algorithmus neu zu starten [135] und der Idee eines dezentralen Ansatzes [1].

Ahmed et. al.[2] stellen eine Möglichkeit zum Neustart vor, in dem die Abstände zwischen den Agenten gemessen werden. Haben sie einen bestimmten Schwellenwert erreicht, so wird der Algorithmus neu gestartet. Worasucheeps Algorithmus [154] überwacht die Verbesserungen des Fitnesswertes in Abhängigkeit von der Geschwindigkeit der Bewegung des Agenten. Wenn dieser Wert einen

Schwellenwert unterschreitet, wird mit einer Exploration angefangen. Eine ganz einfache Möglichkeit wäre auch, den Algorithmus neu zu starten, sobald nach einer vorgegebenen Anzahl an Iterationen keine neue bessere Lösung gefunden wird [115].

Die Idee des dezentralen Ansatzes liegt darin, verschiedene Schwärme zu starten. Das ermöglicht, verschiedene Populationen zu Exploration und Exploitation zu senden, ohne einen zentralen Kontrollmechanismus zu haben. Diese Diversität führt dazu, dass die Möglichkeit des Verlassens lokaler Minima im Vergleich zu den o.g. Ansätzen größer ist. Es gibt mehrere Möglichkeiten, dies zu realisieren. Einige Beispiele sind [36, 99, 152].

3.4.3 Adaptivität

Das Anpassen der Parameter ist eine schwierige Aufgabe, es gibt hier nur zwei bekannte Ansätze, dies zu realisieren. Die einfachste, aber sehr zeitintensive Methode, ist die *Brute-Force Methode*. Hier werden einfach alle möglichen Lösungen ausprobiert. Eberhart [134] und Clerc [25], „Entdecker“ und „Vater“ von PSO, haben sich mehrmals mit diesem Thema beschäftigt.

Die zweite Möglichkeit, die besten Parameter zu finden, ist durch die *metaheuristische Methode*. Mit anderen Worten wird hier das Finden der Parameter in ein Optimierungsproblem umgewandelt und das Lösen des Problems wird einem Algorithmus überlassen [3].

3.5 Abgrenzung zum Stand der Technik

Die bekannten, vorgestellten Techniken zur Parallelisierung fokussieren sich nur auf eine bestimmte Rechnerarchitektur (z. B. GPGPU), auf ein bestimmtes Programmiermodell (z. B. OpenMP) oder nur auf einen bestimmten Algorithmus, welcher parallelisiert wird. Keine der uns bekannten Methoden legt den Fokus auf ein gutes Mapping von Software zu einer vorhandenen Hardware. Weiterhin wird nicht geprüft, ob der untersuchte Algorithmus oder das zu optimierende Problem dafür geeignet sind, eine Parallelisierung durchzuführen. Da Overhead in der parallelen Welt ein Thema ist, muss dieser Punkt berücksichtigt werden. Ein Algorithmus, der

zwar parallel läuft, aber dessen Laufzeit sich im Vergleich zur seriellen Laufzeit verschlechtert, ist weder produktiv noch effizient.

Die schon erwähnte zu frühe Konvergenz kann leider nicht ganz gelöst werden. Es wird nur versucht, sie so weit wie möglich zu vermeiden. Die Ansätze mit einem Neustart des Algorithmus können zwar helfen, aber geben keine Sicherheit darüber, dass es besser wird. Sie funktionieren nach dem Zufallsprinzip. Dagegen sind die dezentralen Lösungen eine bessere Alternative. Diese Arbeit bedient sich eines dieser Ansätze und passt ihn an die geforderten Bedürfnisse an. Weiterhin wird dieser Ansatz parallelisiert, um die vollen Rechnerressourcen zu nutzen.

Die möglichst besten Parameter zu finden, dauert mit den existierenden Methoden einfach zu lange. Der Wertebereich der Parameter liegt oft zwischen $[0, \infty)$ und nicht alle möglichen Lösungen können getestet werden. Mit der zweiten Methode wird das Problem des Findens der Parameter eine Ebene höher verschoben. Die Aufgabe wird zwar von einem anderen Algorithmus übernommen, aber dadurch stellt sich die Frage, wie man den Algorithmus konfigurieren muss, damit er die besten Parameter findet. Daher wird sich die Fachliteratur zunutze gemacht, um eine neue Methode zu entwickeln, welche deutlich weniger lang dauert und nicht die optimalen aber im Mittel sehr gute Ergebnisse liefert.

Bis hier wurden die Probleme einzeln betrachtet. Keiner der vorgestellten Ansätze versucht, alle drei Hauptprobleme zu lösen. Fast alle Ansätze fokussieren sich nur auf ein bestimmtes Problem und es gibt nur wenige, welche zwei Probleme behandeln. Im Gegenzug, fokussiert sich PASS auf die Lösung aller drei Probleme (Parallelität, Stillstand und Adaptivität) gleichzeitig.

Tabelle 3.2 – Vergleich der existierenden Algorithmen mit PASS

Algorithmus	Parallelität	Stillstand	Adaptivität
PASS	+	+	+
MOL [100]	–	–	+
AWPSO [156]	–	–	+
PPSO [21]	+	+	–
PAPSO [92]	+	–	–
Agile Restart [2]	–	+	–
R-PSO [115]	–	+	+

Es wurden einige Algorithmen aus den im Abschnitt 3.1 genannten Algorithmenklassen ausgesucht und in der Tabelle 3.2 aufgelistet. Dort wird gezeigt, welche Probleme sie behandeln und lösen.

3.5.1 Abgrenzung aus der Sicht des Anwenders

Im Abs. 3.5 wurde der Stand der Technik aus der Sicht des Algorithmus betrachtet. Jetzt wird versucht, das zu lösende Problem aus einer anderen Sicht zu betrachten; aus der Sicht des Anwenders.

Folgendes Szenario: Man hat einen *Problemtyp A* und bis jetzt hat man das Problem mit dem CMA-ES Algorithmus gelöst und der Algorithmus liefert sehr gute Ergebnisse. Dann hat man einen neuen *Problemtyp B* zum Lösen bekommen, der sehr unterschiedlich ist und aus einer anderen Domäne kommt. Man versucht das Problem mit dem CMA-ES Algorithmus zu lösen, aber dieser liefert leider sehr schlechte Ergebnisse. Um noch bessere Ergebnisse zu bekommen, macht man sich auf die Suche nach einem neuen Algorithmus. Nach langer Zeit und durch Probieren verschiedener existierender Lösungen, hat man sich entschieden, den DE Algorithmus zu nehmen. Der liefert jetzt die besten Ergebnisse, welche aber noch nicht ganz zufriedenstellend sind. Um die Qualität der Ergebnisse zu steigern, muss man den *Crossover-Rate* Parameter anpassen. Nach der Anpassung und erneuten Tests, stellt man fest, dass die Ergebnisse stimmen, und fährt fort, den Problemtyp B mit dem neuen Algorithmus zu lösen.

Der in dem oben genannten Szenario beschriebene Prozess kann sehr gut durch die Abbildung 3.8 dargestellt werden. In der Wissenschaft und Wirtschaft entstehen sehr oft solche Probleme, die komplex und nicht einfach oder per Hand zu lösen sind. Deshalb wird auf Optimierungsprobleme zurückgegriffen. Die Frage ist aber, welcher Algorithmus genommen werden soll, um möglichst gute Lösungen bezüglich der Laufzeit oder Qualität zu erreichen. Diese generelle Frage wird auch *Algorithm Selection Problem (ASP)* genannt. Leider gibt es für diese Frage keine Lösung (*No Free Lunch Theorem* [153] - s. Def. 3.5.1) und deshalb muss nach einem guten Algorithmus gesucht werden, welcher für das Problem geeignet ist.

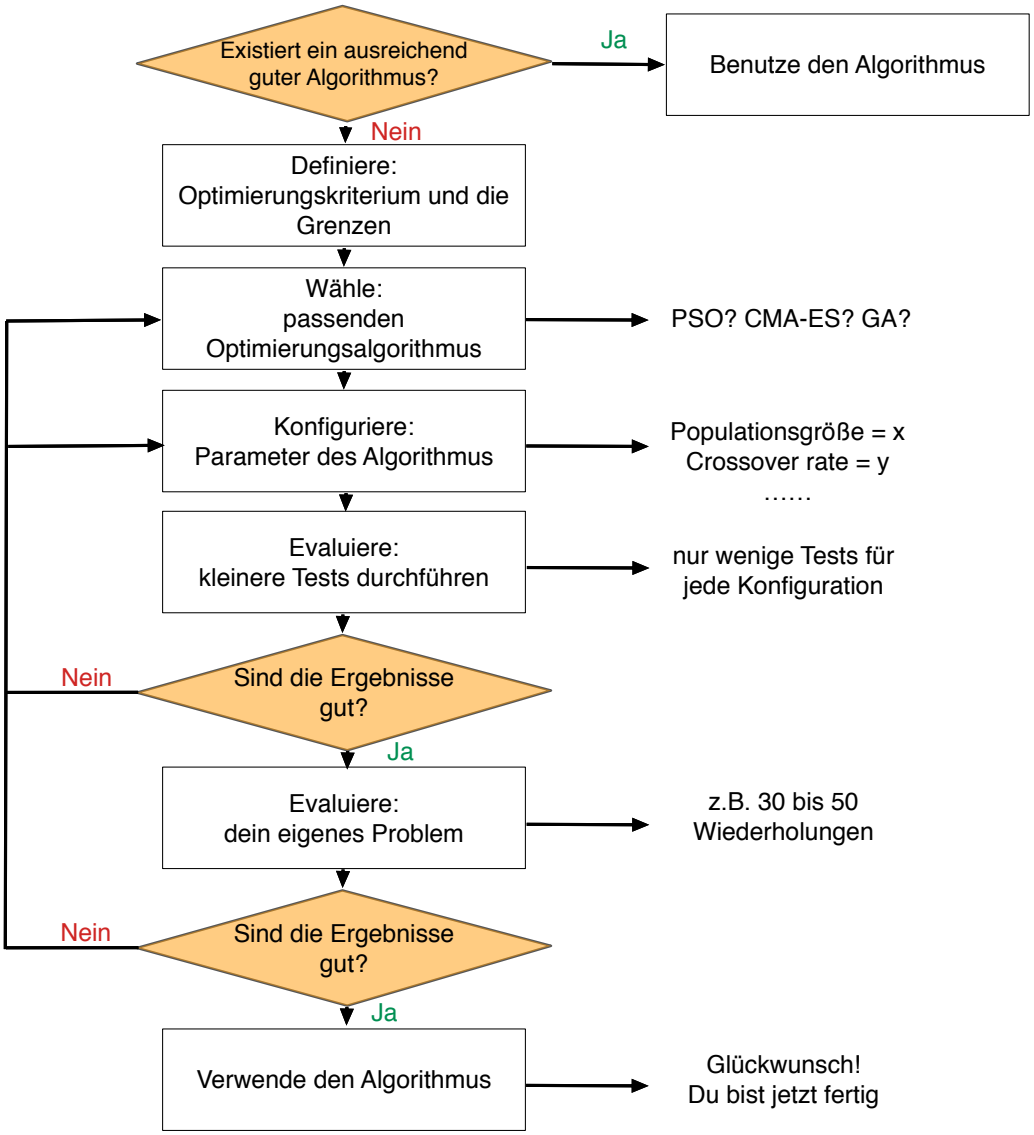


Abbildung 3.8 – Der Ablauf aus der Sicht des Anwenders, sein Problem zu lösen.

Definition 3.5.1 (No Free Lunch Theorem) *Es gibt kein universell gutes Verfahren zur Lösung eines Optimierungsproblems oder zum Abstrahieren von Datensätzen, wenn die Menge aller Probleme bzw. Datensätze betrachtet wird. Ist eine bestimmte Strategie in einer bestimmten Domäne besser als eine andere, so muss sie in einer anderen Domäne schlechter sein. Zusammengefasst: Wenn man die Gesamtheit aller Fälle betrachtet, gibt es keine Strategie, die besser abschneidet als bloßes Raten oder die Zufallssuche.*

PASS versucht durch die integrierte Adaptivität und Automatisierung, besser als der Durchschnitt zu sein. Die Abbildung 3.9 gibt das zu erreichende Ziel von PASS an. Die graue Linie symbolisiert den mathematischen Durchschnitt der Performance über die gesamte Klasse der Problemtypen. Die blaue Linie stellt einen Allgemein-Zweck Algorithmus dar (z. B. Zufallssuche), welcher Ergebnisse liefert, die dem mathematischen Durchschnitt sehr ähneln. Die rote Linie stellt einen spezifischen Algorithmus (z. B. Algorithmen aus der Tab. 3.2) dar, welcher auf eine bestimmte Klasse von Problemen spezialisiert ist und dort sehr gute Ergebnisse liefert, aber für den Rest der Problemtypen leider unterdurchschnittliche Ergebnisse erzielt. Die grüne Linie stellt PASS dar, die im Durchschnitt besser als alle anderen Algorithmen abschneiden soll. Selbstverständlich könnte Algorithmus A oder B für eine bestimmte Domäne besser als PASS sein (s. Abb. 3.9), aber im Durchschnitt ist PASS deutlich besser.

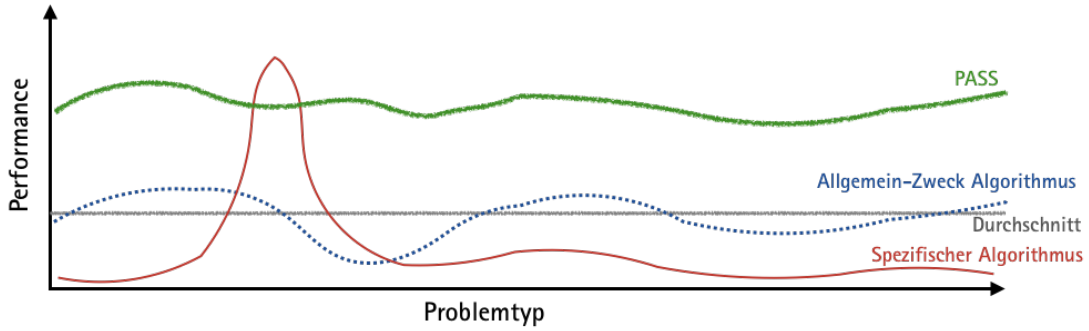


Abbildung 3.9 – Grafische Darstellung des Ziels dieser Dissertation.

3.6 Abgrenzung: Zusammenfassung

Im Abschnitt 3.4 wurden unterschiedliche Techniken und Technologien aus der Fachliteratur aufgezeigt. Die vorgestellten Arbeiten beschäftigen sich mit der Lösung von Teilproblemen, die in ähnlicher Form auch von PASS bearbeitet werden. Die Abgrenzung aus der Sicht des Algorithmus und Anwenders wurde im Abschnitt 3.5 und 3.5.1 erläutert. Jetzt werden alle Punkte kurz in einer Liste zusammengefasst:

1. *Vollständigkeit*: PASS betrachtet alle in Kapitel 1.2 beschriebenen Hauptprobleme der Optimierungsalgorithmen. Während die Arbeiten aus der Fachliteratur

einzelne Probleme mit Erfolg behandeln und lösen, schafft es PASS, alle dieser drei Hauptprobleme zu behandeln und erfolgreich zu lösen.

2. *Automatische Parallelisierung*: PASS nutzt ein Modell, welches in der Lage ist, zu entscheiden, ob eine Parallelisierung sinnvoll ist oder nicht.

3. *Algorithmenanpassung:* Andere Arbeiten ändern den Programmcode bei der Transformation in parallelen Code. Diese sind jedoch allgemein gehalten und nicht speziell auf den zu betrachtenden Algorithmus ausgelegt. PASS ist genau an den spezifischen Algorithmus angepasst und *keine* Programmiercodeanpassung ist notwendig, wenn zwischen parallel und seriell umgeschaltet werden muss.

4. *Rechnerarchitekturanpassung:* Andere Arbeiten sind nur für eine bestimmte Rechnerarchitektur gemacht. PASS funktioniert sowohl für den privaten Rechner, als auch für einen HPCC. Der Endnutzer kann bei der Kompilierung selber bestimmen, welche Eigenschaften der Hardware er benutzen möchte. Diese lassen sich selber ein- und ausschalten, ohne etwas an dem Programmiercode zu ändern.

5. *Offen:* PASS beruht vollständig auf offenen Standards und offener Software. Der Programmiercode ist öffentlich für alle auf Github[3] zugänglich. Es wird nur eine weitere Software-Bibliothek benutzt. Dadurch ergibt sich eine potentiell große Nutzerbasis und die verwendeten Techniken sind sehr einfach nachzuvollziehen.

6. *Erweiterbarkeit:* Alle in PASS verwendeten Klassen sind erweiterbar. Das Framework hat schon einige Optimierungsprobleme und -algorithmen implementiert und bietet die nötigen abstrakten Klassen, um es beliebig zu erweitern. Dazu gibt es noch eine Dokumentation, welche online[4] zur Verfügung steht.

[3] www.github.com/rshuka/PASS
[4] https://rshuka.github.io/PASSDoc/index.html

3.7 Algorithmengrenzen

Im Kapitel 2.4 wurden die Optimierungsprobleme aufgelistet, welche Teil dieser Dissertation sind. In diesem Abschnitt werden die Algorithmen aufgelistet, mit welchen (PASS) funktioniert. In der Regel können die Methoden und Verfahren von PASS mit allen Algorithmen aus der Klasse der populationsbasierten Methoden (s. Abb. 3.1) verwendet werden. Die Details werden nur an einem einzelnen Algorithmus erklärt, aber das Gleiche kann mit den anderen Algorithmen erreicht werden. Dafür müssen sie nur folgende Eigenschaften besitzen:

1. *Populationsbasiert:* Um eine effiziente Parallelisierung des Algorithmus zu ermöglichen, muss dieser eine Population mit mindestens zwei Agenten besitzen.

2. *(Beinahe) Unabhängigkeit bei der Evaluation:* Weiterhin ist eine effiziente Parallelisierung nur gewährleistet, wenn für Agenten die Evaluation des Problems unabhängig von den anderen Agenten ist.

3. *Variablen:* Damit eine bessere Anpassung an das Problem erfolgen kann, müssen die Algorithmen Parameter besitzen, welche die Anpassung des Verhaltens ermöglichen, wie z. B. einem Schalten zwischen Exploration und Exploitation.

3.8 Zusammenfassung

In diesem Kapitel wurde eine Klassifizierung der bestehenden Optimierungsalgorithmen vorgenommen, welche für die Evaluationen verwendet werden. Sie wurden im Detail vorgestellt. Als Nächstes wurden die Probleme in der Forschung aufgelistet, welche solche Algorithmen charakterisieren. Es wurde aus der Fachliteratur gezeigt, wie die Probleme gehandhabt werden und welche Lösungen schon existieren. Dann wurde die Abgrenzung dieser Dissertation vom Stand der Technik dargestellt und der Beitrag dieser Dissertation für die Forschung genauer erläutert. Zusätzlich wurden die Eigenschaften der Algorithmen aufgelistet, für die die Methoden und Techniken dieser Arbeit gelten. Damit wurde eine klare Abgrenzung für die Optimierungsprobleme (s. Abb. 2.4) und für die Optimierungsalgorithmen hergestellt.

3.8 Zusammenfassung

In dem folgenden Kapitel wird die Grundarchitektur von PASS erklärt. Da die Techniken und Methoden nicht für alle Algorithmen der Klasse der populationsbasierten Methoden aufgezeigt werden können, wird nur der PSO als Repräsentant ausgesucht. Dieser wird dann genauer in Einzelheiten erklärt und beschrieben.

4

Die Architektur

Parallel Adaptive Swarm Search

Auf seine eigene Art zu denken ist nicht selbstsüchtig. Wer nicht auf seine eigene Art denkt, denkt überhaupt nicht.

OSCAR WILDE

In diesem Kapitel wird ein Blick auf die Architektur von PASS geworfen. Es wird definiert, welche die Eingaben und Ausgaben des Frameworks sind. Dazu wird PASS aus der Sicht des Endnutzers und aus der Sicht des Entwicklers betrachtet. Als Nächstes wird ein tiefer Einblick in den Standard Particle Swarm Optimization 2011 (SPSO2011) [159] Algorithmus gegeben, da er die Basis für die Evaluation des Frameworks bildet. Die Evaluationsszenarios im darauffolgenden Kapitel werden genau definiert und die ausgewählten Evaluationsprobleme werden vorgestellt. Zum Schluss wird ein Einblick in die Rechnerarchitektur gegeben, in der die Evaluationen stattfinden.

4.1 PASS: Ein Überblick

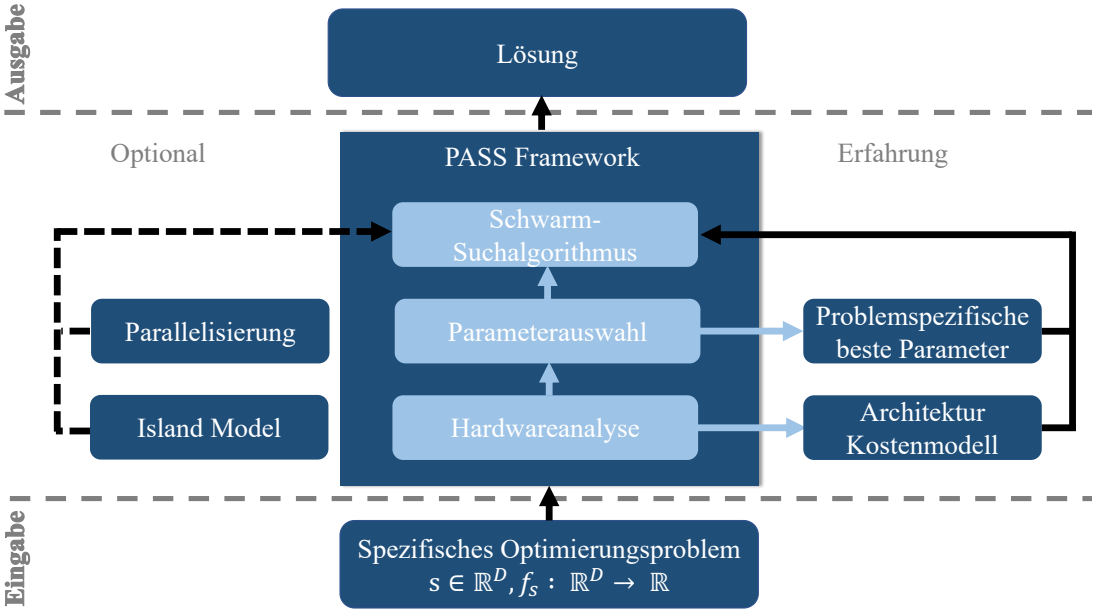


Abbildung 4.1 – Ein Überblick über den gesamten Ansatz von PASS.

Die Abbildung 4.1 veranschaulicht den vorgeschlagenen Ansatz. Als Eingabe dieses Ansatzes dient das spezifische Optimierungsproblem. Das Problem wird immer als Blackbox betrachtet und es bestehen keine weiteren Informationen darüber. Die genaueren Probleme, welche untersucht werden und als Eingabe dienen, sind im Abschnitt 2.5 erwähnt.

Die Lösung ist der Punkt im Suchraum mit den dazugehörigen Koordinaten. Die *Dimension* des Problems variiert von Fall zu Fall. Die Ausgabe hängt von den Abbruchkriterien ab, welche vom Auftraggeber definiert werden. PASS hat folgende Abbruchkriterien implementiert:

1. **Good enough Wert:** Wenn der Auftraggeber einen Zielwert vorgibt, mit dem er zufrieden ist, dann wird der Wert als *good enough* Wert betrachtet und das Framework hört auf zu optimieren, sobald dieser Wert erreicht ist.

2. **Maximale Anzahl an Iterationen:** Wird eine vordefinierte Anzahl von Funktionsiterationen (s. Def. 4.1.1) erreicht, dann wird das Optimieren abgebrochen.

3. **Maximale Anzahl an Funktionsevaluationen:** Wird eine vordefinierte Anzahl von Funktionsevaluationen (s. Def. 4.1.1) erreicht, dann wird das Optimieren abgebrochen.

4. **Maximale Zeitdauer:** Verstreicht eine bestimmte Zeitdauer, dann wird das Optimieren abgebrochen.

Bei den Evaluationen des PASS Frameworks in dem folgenden Kapitel wird genau angegeben, welches Abbruchkriterium benutzt wird.

Definition 4.1.1 (Iteration / Evaluation) *Die Auswertung einer Lösung mittels eines Agenten wird Evaluation genannt. Bei einer Population ist eine Iteration zu Ende, wenn alle Agenten der Population ihre Evaluation ausgeführt haben. Der Zusammenhang kann mathematisch wie folgt ausgedrückt werden:*

*Evaluationen = Anzahl von Iterationen * Populationsgröße*

Das Herz des vorgestellten Ansatzes ist das PASS Framework. Es besteht aus drei Hauptteilen:

1. **Hardwareanalyse:** Das Ziel hier ist, herauszufinden, ob es sich lohnt, eine Parallelisierung zu verwenden. Eine schnellere Ausführung des parallelen Codes im Vergleich mit dem seriellen Code wird angestrebt.

2. **Parameterauswahl:** Das Ziel hier ist, die besten Parameter für das untersuchte Problem herauszufinden. Die Parameterauswahl hat einen großen Einfluss auf die Performance des Algorithmus [134].

3. **Schwarm-Suchalgorithmus:** Der SPSO2011 Algorithmus wird als Beispiellösung genommen und erweitert. Eine genauere Erklärung findet im Abschnitt 4.3 statt.

Eine genauere Erklärung der oben genannten Schritte folgt in dem nächsten Kapitel.

Sowohl die *Hardwareanalyse* als auch die *Parameterauswahl* erzeugen ein Ergebnis bzw. eine Datei, welche in Zukunft verwendet werden kann. Das erste Ergebnis

ist ein Architektur-Kostenmodell und das Zweite ein Ergebnis mit den besten Parametern für das aktuelle Problem. Die beiden Schritte werden jeweils nur einmal gemacht. Die Hardwareanalyse wird nur durchgeführt, wenn die Architektur geändert wird, in der PASS läuft, und die Parameterauswahl wird nur durchgeführt, wenn ein neues zu optimierendes Blackbox-Problem auftritt.

Die Parallelisierung und das Island Model sind optional und können nach Bedarf aktiviert werden. Genauere Details folgen im nächsten Kapitel.

4.2 Methodik vs. Softwareimplementierung

Die vorgestellte Methodik dieser Arbeit ist für die Klasse der populationsbasierten Algorithmen gültig. Welche Eigenschaften diese Algorithmen erfüllen müssen, wurde im Abschnitt 3.5 erklärt. Die Methodik wird dann durch ein Werkzeug, das PASS Framework, implementiert. Die Implementierung folgt nur für einen Repräsentanten der Klasse der populationsbasierten Algorithmen, den SPSO2011 (s. Ab. 4.3).

Das Framework kann von jedem verwendet werden. Die einzige benötigte Eingabe ist das Optimierungsproblem. Dafür bietet das Framework eine Schnittstelle. Der Endnutzer muss nur das zu optimierende Problem liefern und das Framework nach seinen Wünschen konfigurieren. Keine weiteren Kenntnisse sind erforderlich und alles läuft automatisch.

Möchte man aber die vorgestellte Methodik auf einen anderen Algorithmus anwenden, ist Expertenwissen gefragt. Der Entwickler in dem Fall muss manche Sachen manuell durchführen. Eine effiziente Parallelisierung des Algorithmus ist nur durch manuelle Eingriffe möglich und Expertenwissen über die Parameter des Algorithmus ist gefragt, um die Suchräume zu begrenzen. In diesem Fall handelt es sich um eine semi-automatische Lösung. Die Methodik, die Schnittstellen und die dazugehörigen Verfahren werden mit PASS geliefert. Dazu steht noch ein dokumentiertes Application Programming Interface (API) zur Verfügung. PASS wird als freie Software unter der MIT Licence[5] geliefert.

[5] https://opensource.org/licenses/mit-license.php

4.3 PSO Algorithmus

Wie schon oben erwähnt, wird die Methodik anhand eines Algorithmus der populationsbasierten Klasse vorgestellt. Hier wurde der PSO Algorithmus ausgesucht. In den folgenden Abschnitten wird ein tieferer Einblick in den Algorithmus gegeben und die wichtigsten Teile des Algorithmus werden vorgestellt.

4.3.1 Entwicklung

Die ersten Versuche, das soziale Verhalten von Schwärmen auf einem Algorithmus nachzubilden, wurden durch Kennedy und Eberhart im Jahr 1995 unternommen [88]. Die erste Idee war, eine Gruppe von Agenten zufällig auf den Suchraum zu verteilen. Diese Agenten haben zwei Hauptstrategien für die Suche verwendet: 1) Die Anpassung der Geschwindigkeit anhand des Nachbarn und 2) die sogenannte *Craziness* (dt. Verrücktheit). Bei jeder Iteration wurden die Nachbarn eines Agenten und deren Geschwindigkeit definiert. Das Problem mit diesem Ansatz war, dass sich die Agenten dann nur in eine Richtung orientiert haben und es gab keine Möglichkeit, die Richtung zu ändern oder umzukehren. Um solches Verhalten zu verhindern, wurde eine stochastische Variable eingeführt, die *Craziness*. Bei jeder Iteration wurden zufällig verschiedene Geschwindigkeiten ausgewählt und mit einer Änderung ergänzt. Durch diese Änderung ähnelt der Algorithmus mehr einem sozialen Verhalten als am Anfang. Die oben genannten Beobachtungen weisen auf eines der notwendigsten Merkmale von PSO hin, das auf seine scheinbar unveränderliche, nicht-deterministische Natur hinweist: Die Einbeziehung der Zufälligkeit.

Kennedy und Eberhart leiteten den nächsten Schritt ein, indem sie den Begriff *roost* (dt. Schlafstelle) von einem Platz, den die Agenten früher kannten, in *food* (dt. Nahrung) änderten; einem Platz, den die Vögel jetzt suchen müssen. Durch diese Änderung wurde der Algorithmus von einer sozialen Simulation zu einem Optimierungsalgorithmus. Die Grundidee lag darin, dass die Agenten (Vögel) einen unbekannten guten Ort im Suchraum (Nahrungsquelle) suchen müssen. Die Suche findet statt, indem die Agenten miteinander kooperieren. Jeder Agent war in der Lage, sich seine beste Position und die beste Position des gesamten Schwarmes zu merken.

Angesichts der Position $\vec{X}_i = (x_{i1}, x_{i2}, \ldots, x_{iD}) \in S$ und der Geschwindigkeit $\vec{V}_i = (v_{i1}, v_{i2}, \ldots, v_{iD}) \in \mathbb{R}^D$ des $i-ten$ Agenten und der besten persönlichen Position des Agenten $\vec{P}_i = (p_{i1}, p_{i2}, \ldots, p_{iD}) \in S$ und der besten Position in der Nachbarschaft $\vec{L}_i = (l_{i1}, l_{i2}, \ldots, ld_{iD}) \in S$, kann die nächste Position $\vec{X}_i^{t+1}$ durch folgende Gleichungen definiert werden:

$$\vec{V}_i^{t+1} = \vec{V}_i^t + c_1\vec{U}_1^t \otimes (\vec{P}_i^t - \vec{X}_i^t) + c_2\vec{U}_2^t \otimes (\vec{L}_i^t - \vec{X}_i^t) \tag{4.1}$$

$$\vec{X}_i^{t+1} = \vec{X}_i^t + \vec{V}_i^{t+1} \tag{4.2}$$

wo $i \in \{1, 2, \ldots, Pop\}$, mit Pop gleich der Schwarmgröße und $t \in \{1, 2, \ldots, T\}$ mit T gleich der maximalen Iterationenanzahl. Die Variablen c_1 und c_2 werden respektiv lokale Anziehungskraft und globale Anziehungskraft genannt. Die Variablen $\vec{U}_1$ und $\vec{U}_2$ sind uniform verteilte Zufallsvektoren aus $[0, 1]^D$. Das Symbol $\otimes$ ist das elementweise Produkt.

In der ersten Version von PSO [88] wurden die Anziehungskräfte c_1 und c_2 mit dem Wert 2 versehen. Diese zwei Variablen sind die wichtigsten in dem Algorithmus, weil sie die Beziehung zwischen Exploration und Exploitation genau bestimmen. Ein hoher Wert von c_2 führt dazu, dass sich die Agenten in die Richtung des besten globalen Wertes der Nachbarschaft bewegen, und ein hoher Wert von c_1 führt zu einer schnellen lokalen Konvergenz. Später führte Eberhart et al. [45] eine neue Variable $\vec{V}_{max}$ ein, welche dazu dienen sollte, die Agenten am Verlassen des Suchraums zu hindern. Weitere Untersuchungen über das Verhalten des Algorithmus wurden unternommen und Shi und Eberhart [133] sind zu dem Ergebnis gekommen, dass der erste Term (vorherige Geschwindigkeit) in der Gleichung 4.1 eine wichtige Rolle in dem Zusammenhang zwischen Exploration und Exploitation spielt. Wird dieser Term weg gelassen, dann sind die Agenten nicht in der Lage, den Bereich zu verlassen, in dem sie sich befinden, und der Suchraum wird mit der Zeit immer kleiner. Das ist äquivalent mit einer lokalen Suche (Exploration). Im Gegensatz werden die Partikel dazu tendieren, den Suchraum in Richtung des besten Wertes der Nachbarschaft abzutasten, wenn die vorherige Geschwindigkeit einen sehr hohen Wert aufweist. Die Agenten werden dann überwiegend versuchen, unbesuchte Regionen zu untersuchen. Das wäre mit einer globalen Suche vergleichbar (Exploitation). Da wir aber beide Verhalten brauchen, um gute Ergebnisse zu erzielen, muss ein Gleichgewicht herrschen. Deshalb wurde in der Gleichung 4.1 die Variable *inertia weight* $w \in \mathbb{R}$ eingeführt:

$$\vec{V}_i^{t+1} = w\vec{V}_i^t + c_1\vec{U}_1^t \otimes (\vec{P}_i^t - \vec{X}_i^t) + c_2\vec{U}_2^t \otimes (\vec{L}_i^t - \vec{X}_i^t) \tag{4.3}$$

Es gibt verschiedene Arbeiten, welche sich mit dem Einfluss von *inertia weight* beschäftigt haben. Das hat dazu geführt, dass verschiedene Abwandlungen des PSO Algorithmus eingeführt wurden [9].
Auch nach den verschiedenen Änderungen und Anpassungen am PSO gab es ein Problem, welches noch nicht gelöst wurde. Durch die Gleichung 4.3 wird das Aktualisieren der Geschwindigkeit für jede Dimension getrennt durchgeführt und das führt dazu, dass der Algorithmus sehr abhängig vom Koordinatensystem ist (Rotationsempfindlichkeit). Es war gut zu erkennen, dass die Achsen sehr bevorzugt wurden. Das hat dazu geführt, dass Probleme, welche ihre Lösung beim Punkt $(0, 0, \ldots 0)$ hatten, sehr schnell gelöst wurden. Um dieses Verhalten zu verhindern, hat Clerc [159] die Gleichung 4.3 modifiziert und eine neue Variable hinzugefügt. Diese Variante des Algorithmus, welche im Jahr 2013 auf der CEC-Konferenz [73] vorgestellt wurde, ist die letzte Standardvariante des PSO Algorithmus und dient als Referenz für die zukünftigen Abwandlungen des PSO. Die Variante wird auch SPSO2011 genannt und ist die Basis von PASS.

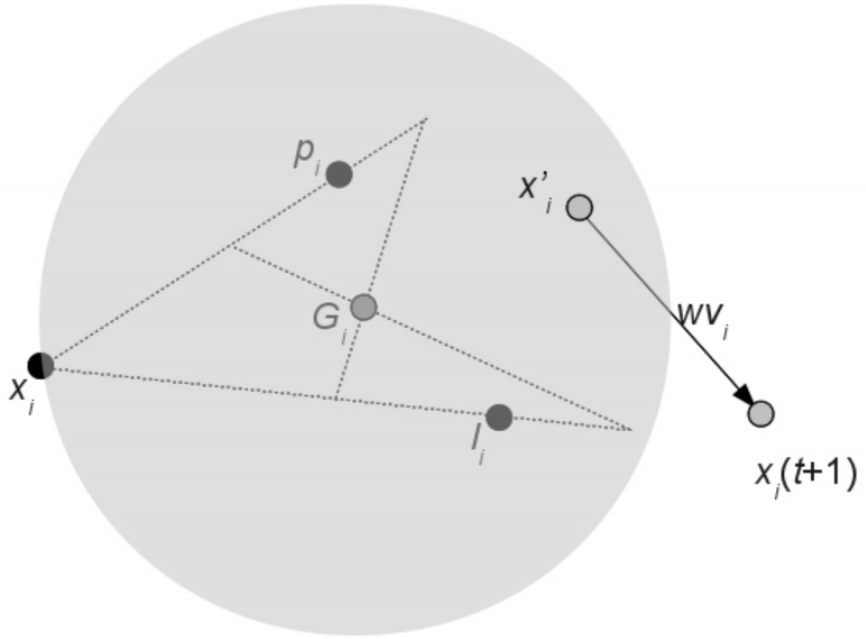


Abbildung 4.2 – In den SPSO2011 wird der Punkt x' innerhalb der Hypersphere H_i zufällig ausgesucht.

Die Funktionsweise der SPSO2011 ähnelt anderen, früheren Implementierungen. Auch hier besteht der Schwarm-Algorithmus aus einer Menge von Agenten, die in einem Suchraum mit D Dimensionen nach der optimalen Lösung suchen. Alle Bewegungen des Agenten im Suchraum sind von der eigenen Beschleunigung abhängig. Diese Beschleunigung wird unter anderem von der persönlichen Anziehungskraft der besten Lösung des Agenten $\vec{P}_i$ und der globalen Anziehungskraft der insgesamt besten in der Nachbarschaft gefundenen Lösung $\vec{L}_i$ beeinflusst. Der SPSO2011 ist im Vergleich zu den Vorgängern gegenüber Rotation unempfindlich. Für jeden Agenten in jedem Zeitstempel wird ein Gravitationspunkt $\vec{G}_i$ deklariert (s. Gl. 4.6), welcher sich zwischen drei Punkten befindet: Der aktuellen Position $\vec{X}_i^t$, einem kleineren Punkt jenseits von der besten persönlichen Lösung $\vec{p}_i^t$ und einem Punkt jenseits von der besten gefundenen Lösung $\vec{l}_i^t$ in der Nachbarschaft (s. Abb. 4.2) .

$$\vec{p}_i^t = \vec{X}_i^t + c_1 \vec{U}_1^t \otimes (\vec{P}_i^t - \vec{X}_i^t) \tag{4.4}$$

$$\vec{l}_i^t = \vec{X}_i^t + c_c \vec{U}_2^t \otimes (\vec{L}_i^t - \vec{X}_i^t) \tag{4.5}$$

$$\vec{G}_i^t = \frac{(\vec{X}_i^t + \vec{p}_i^t + \vec{l}_i^t)}{3} \tag{4.6}$$

Hier wird dann der nächste Punkt x' aus der Hypersphere H_i mit Mittelpunkt $\vec{G}_i$ und Radius $||\vec{G}_i^t - \vec{X}_i^t||$ zufällig ausgesucht.

Dadurch wird die Geschwindigkeit mit der folgenden Formel vermittelt:

$$\vec{V}_i^{t+1} = w\vec{V}_i^t + H_i(\vec{G}_i^t, ||\vec{G}_i^t - \vec{X}_i^t||) - \vec{X}_i^t \tag{4.7}$$

während die Position mit der Gleichung 4.2 berechnet wird.

Der SPSO2011kann wie folgt zusammengefasst werden [159]:

```
    Eingababe SN: Populationsgröße

1   for i=1 to SN do
2     Initialise particle's positions $\vec{X}_i$ and velocities $\vec{V}_i$
3     Initialise personal/previous best, $\vec{P}_i$, $\vec{L}_i$, $\vec{G}$
4   end
5   while ¬terminationCriterion() do
6     for i=1 to SN do
7     Update particle's velocity using eq.4.2
8     Update particle's position using eq.4.7
9     if $f(\vec{X}_i) < f(\vec{P}_i)$ then {minimization of f}
10      Update particle's best–known position $\vec{P}_i = \vec{X}_i$
11      if $f(\vec{P}_i) < f(\vec{L}_i)$ then {minimization of f}
12        Update the neighbourhood's best–known position $\vec{L}_i = \vec{P}_i$
13      end
14    end
15  end
16 end
```

Pseudocode 4.1 – SPSO2011

4.3.2 Topologie

Die Topologie ist definiert als die Menge aller Partikel, mit denen jeder Agent in der Lage ist, Informationen auszutauschen. Die ersten Implementierungen von PSO haben eine globale Topologie gehabt. In dieser Art der Nachbarschaft kann jeder Agent mit jedem kommunizieren. Diese Art der Kommunikation ist nicht immer zwingend notwendig und andere Arbeiten haben gezeigt, dass eine andere Art der Kommunikation effizienter sein kann [53, 106]. Die Abbildung 4.3 zeigt mögliche Topologien [106]. Es können aber auch Topologien selbst definiert und benutzt werden. In der Richtung gibt es keine Einschränkungen.

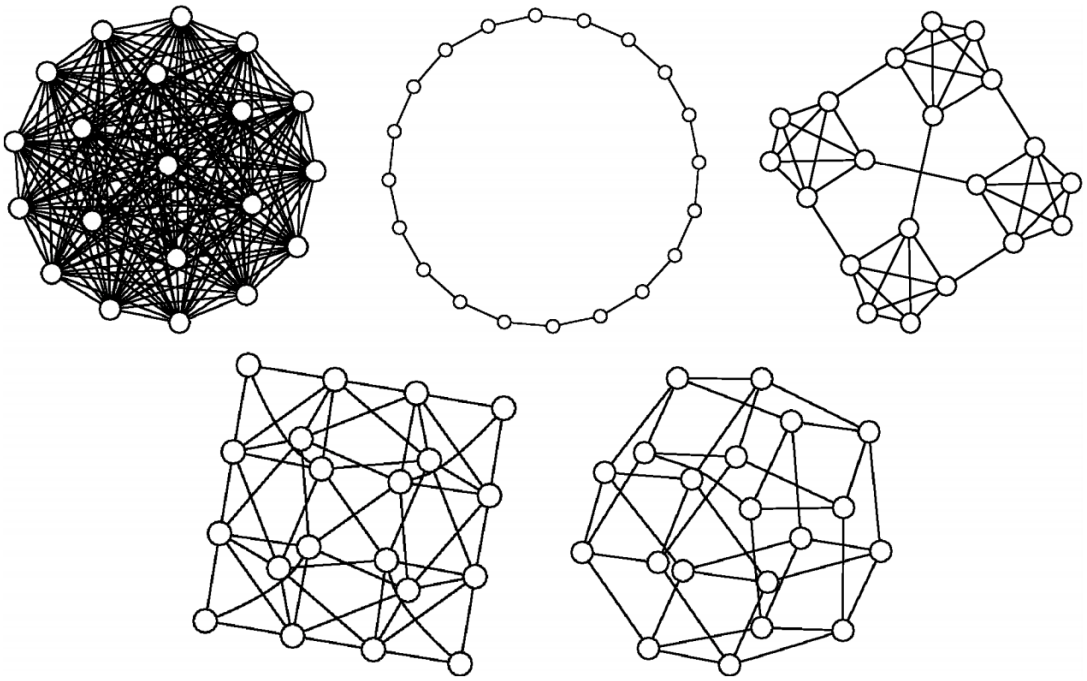


Abbildung 4.3 – Einige mögliche Nachbarschaftsbeziehungen für den PSO [106].

Die Art der Topologie spielt eine Rolle für das Verhalten des Algorithmus. Durch verschiedene Nachbarschaften kann der Fluss von Informationen in einem Schwarm beeinflusst werden und somit auch die Konvergenzrate und das Verhältnis zwischen Exploration oder Exploitation. Eine globale Topologie kann, z. B. zu einer frühen Konvergenz führen, da alle Agenten mit den besten Agenten verbunden sind und von diesen beeinflusst werden. Dadurch ist der Schwarm nicht in der Lage, andere interessante Suchräume zu erforschen, und wird in einem lokalen Minimum gefangen bleiben.

Die Ring-Topologie ist die gegenläufige Alternative zur globalen Topologie, in der die Agenten mit nur zwei anderen Agenten kommunizieren. Diese Art der Nachbarschaft ermöglicht einen langsamen Austausch der Informationen und somit eine langsamere Konvergenz, da die Information lange braucht, um alle Agenten zu erreichen.

Die anderen Topologien bewegen sich zwischen diesen beiden. Welche Topologie die beste ist, ist problemabhängig.

Der SPSO2011 hat eine zufällige Topologie [23]. Nach jedem Iterationsschritt bleibt die Topologie erhalten, wenn eine bessere Lösung gefunden wird, sonst wird eine neue zufällige Topologie geworfen. Die Initialisierung wird durch eine Variable ($\boldsymbol{np}$) beeinflusst. Der Wertebereich dieser Variable ist $[0,1]$, wo die 0 gar keine Kommunikation bedeutet und die 1 eine globale Topologie ermöglicht.

4.3.3 Begriffserklärungen

Nachdem die Entwicklung des PSO erläutert wurde, folgt eine Zusammenfassung und Erklärung der wichtigsten Begriffe, welche für das laufende Kapitel relevant sind:

Partikel: Bis jetzt wurde der Term *Agent* benutzt. Für den PSO sagt man aber Partikel statt Agent. Jeder Partikel i repräsentiert ein Individuum des Schwarms. Die Partikel besitzen eine Variable $\vec{P}_i$, welche die Position des Partikels im Suchraum widerspiegelt. Außerdem hat der Partikel eine Geschwindigkeit $\vec{V}_i$, welche die Bewegung durch den Suchraum beschreibt.

Schwarm: Die Größe des Schwarms Pop besteht aus einer vordefinierten Anzahl von Partikeln.

Beschleunigung: Repräsentiert die Beschleunigung, mit der ein Partikel sich im Suchraum bewegt. w ist die Beschleunigungskonstante (engl. inertia weight).

$\vec{P}_i$: Repräsentiert die beste gefundene persönliche Position des Partikels i. Der Parameter c_1 ist die lokale Anziehungskraft.

$\vec{L}_i$: Repräsentiert die beste gefundene Position des Partikels i in der Nachbarschaft. Der Parameter c_2 ist die globale Anziehungskraft.

Topologie: Die Topologie ist definiert als die Menge aller Partikel, mit denen jeder Partikel in der Lage ist, Informationen auszutauschen.

np: Der Parameter np bestimmt die Art der Topologie.

4.4 Evaluationen

Um die Effizienz eines Algorithmus mit einem anderen Algorithmus zu vergleichen, werden zwei Kriterien berücksichtigt: (1) Die Qualität der Lösung und (2) wie schnell diese Lösung gefunden wurde. Um PASS zu untersuchen, wurden zwei Problemklassen implementiert. Die erste Klasse gehört zu den Black-Box Optimization Benchmark (BBOB) Problemen [33]. Diese werden benutzt, um die Korrektheit der Methodik zu untersuchen. Die zweite Klasse ist aus der realen Welt. Hier werden Probleme von Missionen im Weltall behandelt, in denen die interplanetaren Bahnkurven bestimmt werden müssen.

Die Evalationen folgen in zwei Schritten: (1) Durch die BBOB Probleme werden die Einzelteile von PASS evaluiert und (2) durch die Space Mission Probleme wird PASS als Ganzes mit den im Kapitel 3 vorgestellten Algorithmen verglichen.

4.4.1 Black-Box Optimization Benchmark

Die BBOB sind aus [33] und die Tabelle 4.1 gibt die Funktionen, deren Grenzen und die jeweiligen Postionen der **Minima** an. Bei der Auswahl der Probleme wurde versucht, alle möglichen Charakteristiken abzudecken (s. Abs. 2.4). Die Probleme werden in drei verschiedene Gruppen klassifiziert.

Tabelle 4.1 – BBOB Probleme aus [33].

Benchmark-Funktion	Grenzen	Minimum	Gruppe
F1 $= -20exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}) - exp(\frac{1}{D}\sum_{i=1}^{D} cos(2\pi x_i)) + 20 + e$	$[-32; 32]^D$	0	3
F2 $= \sum_{i=1}^{D} x_i^2$	$[-5,12; 5,12]^D$	0	1
F3 $= 1 + \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} cos(\frac{x_i}{\sqrt{i}})$	$[-600; 600]^D$	0	2
F4 $= 10D + \sum_{i=1}^{D}(x_i^2 - 10cos(2\pi x_i))$	$[-5,12; 5,12]^D$	0	3
F5 $= \sum_{i=1}^{D-1}(100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2)$	$[-2,048; 2,048]^D$	0	2
F6 $= \sum_{i=1}^{D}(-x_i sin(\sqrt{\lvert x_i \rvert})) + 418.982887 \cdot D$	$[-500; 500]^D$	0	3
F7 $= \frac{1}{2}\sum_{i=1}^{D}(x_i^4 - 16x_i^2 + 5x_i)$	$[-5; 5]^D$	$-39,16599 \cdot D$	2
F8 $= \sum_{i=1}^{D} \lvert x_i \rvert^{(i+1)}$	$[-1, 1]^D$	0	1

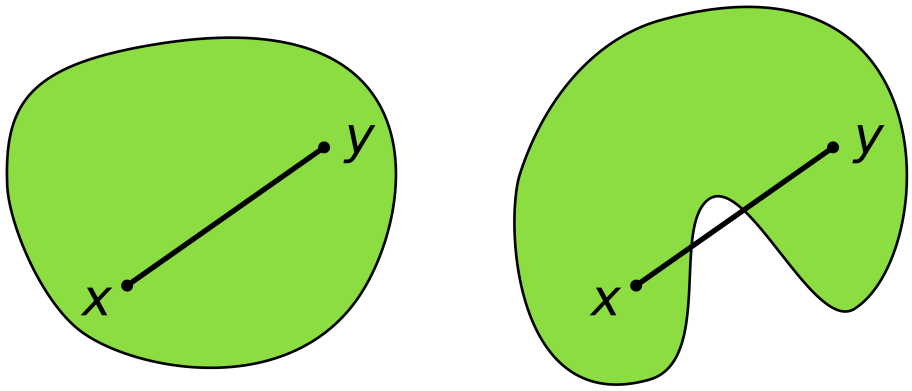
Die erste Gruppe beinhaltet Probleme, die leicht zu lösen sind. Dagegen sind in der dritten Gruppe Probleme beinhaltet, welche sehr schwer zu lösen sind, und die zweite Gruppe ist von Problemen repräsentiert, welche zwischen der ersten und dritten Gruppen liegen.

Probleme, welche *unimodal* und *konvex* (s. Def. 4.4.1) sind, werden in die erste Gruppe eingeordnet. Probleme, welche *multimodal* und *nichtkonvex* sind, werden in die dritte Gruppe eingeordnet.

Definition 4.4.1 (Konvexe Menge) *Eine Menge heißt konvex, wenn für je zwei beliebige Punkte aus dieser Menge, auch stets deren Verbindungsstrecke ganz in der Menge liegt [102]. Die Abbildung 4.4 stellt eine konvexe und eine nichtkonvexe Menge dar.*

Die ausgesuchten Probleme (s. Tab. 4.1) sind folgende:

F1 - Ackley Funktion: Die Funktion ist nichtkonvex und multimodal. Sie wird durch eine sehr hohe Anzahl an lokalen Minima charakterisiert. Das globale Minimum $f(x) = 0$ liegt bei dem Punkt $x_i = 0$ für $i = 1, ..., D$. Die Ackley Funktion gehört zu der Klasse der schweren Probleme.



(a) Eine konvexe Menge **(b)** Eine nichtkonvexe Menge

Abbildung 4.4 – x und y sind zwei zufällige Punkte aus einer Menge.

F2 - Sphere Funktion: Die Funktion ist konvex, stetig und unimodal. Das globale Minimum $f(x) = 0$ liegt bei dem Punkt $x_i = 0$ für $i = 1, ..., D$. Die Sphere Funktion gehört zu der Klasse der einfachen Probleme.

F3 - Griewank Funktion: Die Funktion ist nichtkonvex und stetig. Sie wird durch viele lokale Minima charakterisiert. Die lokalen Minima haben eine regelmäßige Verteilung und sind weit verbreitet im Suchraum. Je höher die Dimension der Funktion ist, desto höher ist die Anzahl der Minima. Das globale Minimum $f(x) = 0$ liegt bei dem Punkt $x_i = 0$ für $i = 1, ..., D$. Die Griewank Funktion gehört zu der Klasse der mittel-schweren Probleme.

F4 - Rastrigin Funktion: Die Funktion ist konvex, stetig, separierbar und multimodal. Die Funktion basiert auf der Sphere Funktion mit der Zugabe vom Kosinus, um möglichst viele lokale Minima zu erzeugen. Die Minima sind regelmäßig verteilt. Das globale Minimum $f(x) = 0$ liegt bei dem Punkt $x_i = 0$ für $i = 1, ..., D$. Die Rastrigin Funktion gehört zu der Klasse der schweren Probleme.

F5 - Rosenbrock Funktion: Die Funktion ist konvex, stetig, nichtseparierbar und multimodal. Das globale Optimum liegt in einem langen, schmalen, parabolisch geformten flachen Tal. Dieses Tal zu finden, ist relativ leicht, dagegen ist die Konvergenz zu dem globalen Minimum sehr schwer. Das globale Mini-

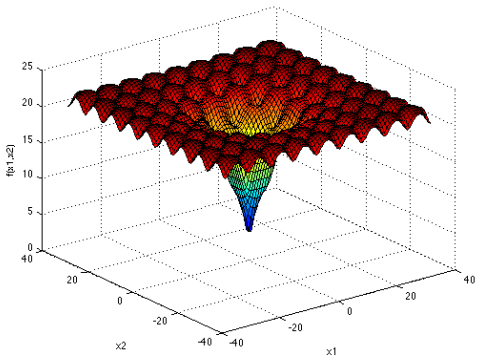
mum $f(x) = 0$ liegt bei dem Punkt $x_i = 1$ für $i = 1, ..., D$. Die Rosenbrock Funktion gehört zu der Klasse der mittel-schweren Probleme.

F6 - Schwefel Funktion: Die Funktion ist nichtkonvex, nichtseparierbar und multimodal. Die Funktion ist durch viele lokale Minima charakterisiert, welche entfernt voneinander und über den gesamten Suchraum verteilt sind. Das globale Minimum $f(x) = 0$ liegt bei dem Punkt $x_i = 420,9687$ für $i = 1, ..., D$. Die Schwefel Funktion gehört zu der Klasse der schweren Probleme.

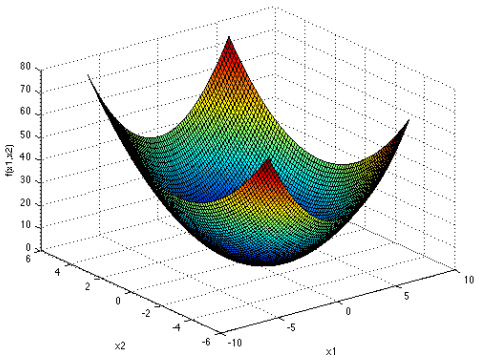
F7 - Styblinski-Tang Funktion: Die Funktion ist nichtkonvex, stetig und multimodal. Die Funktion ist durch viele lokale Minima charakterisiert, welche entfernt voneinander und über den gesamten Suchraum verteilt sind. Das globale Minimum $f(x) = -39,16599 \cdot D$ liegt bei dem Punkt $x_i = -2,903534$ für $i = 1, ..., D$. Die Styblinski-Tang Funktion gehört zu der Klasse der mittelschweren Probleme.

F8 - Sum of Different Powers Funktion: Die Funktion ist nichtkonvex, stetig und unimodal. Eine minimale Änderung einer der Eingaben hat erhebliche Auswirkungen auf den Wert der Funktion. Das globale Minimum $f(x) = 0$ liegt bei dem Punkt $x_i = 0$ für $i = 1, ..., D$. Die Sum of Different Powers Funktion gehört zu der Klasse einfachen Probleme.

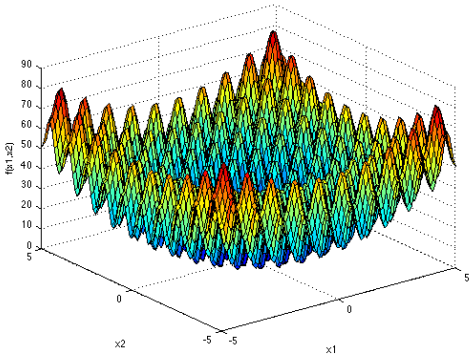
Die BBOB werden sowohl in den niedrigeren Dimensionen (z. B. 2 oder 5) als auch in den höheren Dimensionen (z. B. 20 oder 50) untersucht. Die Abbildung 4.5 zeigt sechs Probleme in der dritten Dimension grafisch.



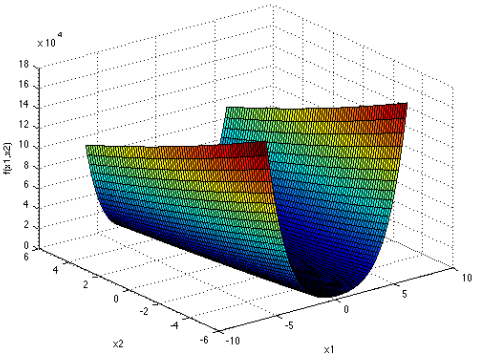
(a) Ackley Funktion F1: nichtkonvex und multimodal (Gruppe 3).



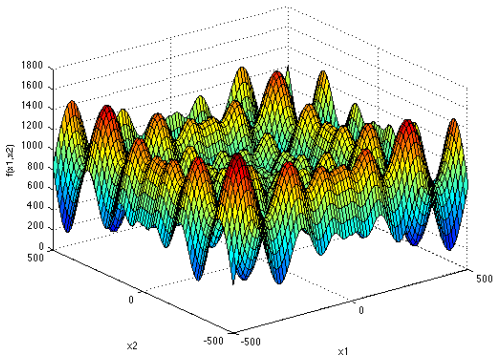
(b) Sphere Funktion F2: konvex, stetig und unimoda (Gruppe 1)l.



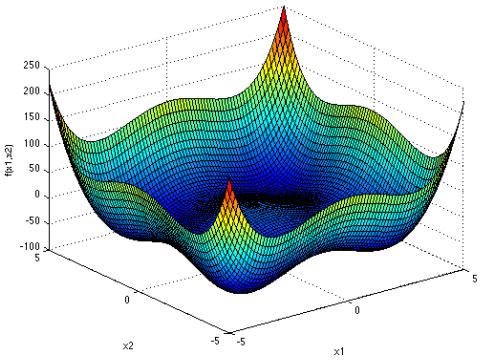
(c) Rastrigin Funktion F4: konvex, stetig, separierbar und multimodal (Gruppe 3).



(d) Rosenbrock Funktion F5: konvex, nichtseparierbar und multimodal (Gruppe 2).



(e) Schwefel Funktion F6: nichtkonvex, nichtseparierbar und multimodal (Gruppe 3).



(f) Styblinski-Tang Funktion F7: nichtkonvex, stetig und multimodal (Gruppe 2).

Abbildung 4.5 – Die dreidimensionale Darstellung von sechs BBOB Problemen [142].

Die BBOB werden benutzt, um die Korrektheit der Methodik zu untersuchen. Wie die Experimente genau aufgebaut sind, wird in den jeweiligen Abschnitten definiert.

4.4.2 Space Mission

Die Advanced Concept Team (ACT) der European Space Agency (ESA) hat seit dem Jahr 2005 eine Datenbank für die Global Trajectory Optimization (GTOP) Probleme veröffentlicht [137] und ermutigt Wissenschaftler aus dem Gebiet der Raumfahrttechnik und benachbarten Gebieten, die Probleme zu untersuchen und Lösungen für die Benchmarks einzureichen. Diese Benchmarks sind als GTOP Datenbank [42] bekannt und repräsentieren Trajektorienmodelle von interplanetaren Bahnkurven verschiedener Raumfahrtmissionen aus der realen Welt. Aktuell besteht diese Datenbank aus acht verschiedenen Benchmark Problemen, welche als ein einzelnes Problem veröffentlicht wurden. Die Ausnahme bildet das Tandem Benchmark, das aus 50 verschiedenen Instanzen besteht.

Die Tabelle 4.2 gibt einen Überblick der Probleme und die Anzahl der Variablen, welche jedes Problem besitzt. Dazu wird die Anzahl der Einreichungen gezeigt und die Zeit zwischen der ersten und letzten Einreichung.

Die GTOP Benchmark Probleme sind reale Blackbox-Probleme und hochgradig nichtlinear. Sie sind sehr schwer zu lösen, auch wenn die Anzahl der Variablen nicht so hoch ist. Die Schwierigkeit des Lösens dieser Probleme kann indirekt auch durch die Zeitspanne der Einreichungen ausgedrückt werden. Cassini1 ist das einfachste Problem, während Messenger (full) das schwierigste Problem ist. Wie aus der Tabelle 4.2 zu lesen ist, hat es mehr als fünf Jahre zwischen der ersten und letzten Lösung gedauert.

Die GTOP Benchmark Probleme sind so schwierig, dass sich die meisten Publikationen nur mit einem oder zwei dieser Probleme beschäftigen [10, 56]. Nur M. Schlueter [129] und Stracquadanio [140] haben alle Probleme der Datenbank untersucht. Dazu muss gesagt werden, dass Teams wie das von M. Schlueter, seit ca. 10 Jahren an der Thematik arbeiten.

Für die Evaluation von PASS wurden insgesamt vier der sieben einzelnen Probleme untersucht. Das einfachste (Cassini1) und das schwierigste Problem (Messenger full), sowie GTOC1 und Rossetta wurden gewählt und näher betrachtet.

Tabelle 4.2 – GTOP Benchmark Probleme

Benchmark	Variablen	Anzahl der Einreichungen	Zeit zwischen der ersten und letzten Einreichung
Messenger(reduced)	18	3	11 Monate
Messenger(full)	26	10	63 Monate
Cassini2	22	7	14 Monate
Rosetta	22	7	6 Monate
Sagas	12	1	-
GTOC1	8	2	24 Monate
Cassini1	6	3	6 Monate
Tandem	18	-	-

4.4.3 Messenger (full mission) Benchmark Problem

In diesem Abschnitt wird exemplarisch ein Problem aus der GTOP Datenbank näher betrachtet. Alle Probleme sind gleich aufgebaut und einzelne Informationen können der ESA-Webseite [42] entnommen werden.

MESSENGER (eng. Mercury Surface, Space Environment, Geochemistry and Ranging) war eine NASA-Raumsonde, welche den Planeten Merkur erforschte. Die Abbildung 4.6 zeigt den Flug der Raumsonde von der Erde bis zum Planeten Merkur. Die Raumsonde musste während ihres Fluges einige Gravity Assist (s. Def. 4.4.2) Manöver durchführen, um den Planeten Merkur zu erreichen.

Die Sequenz der Manöver für diese Mission ist folgende: *Erde-Venus-Venus-Merkur-Merkur-Merkur-Merkur*, wobei das erste Element der Startplanet und das letzte Element der Zielplanet ist. Das Ziel dieses Benchmark Problems ist die Minimierung der gesamten Geschwindigkeitsänderungen ΔV während der Mission. In anderen Worten, wird versucht den Treibstoffverbrauch auf ein Minimum zu reduzieren.

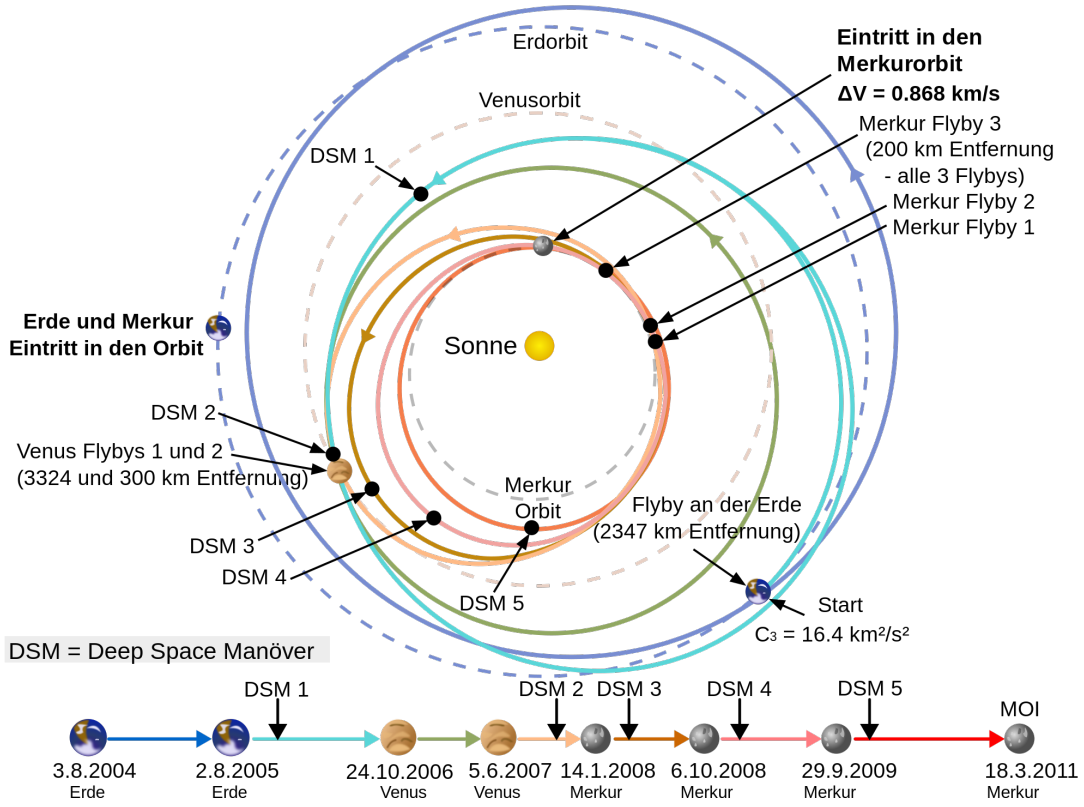


Abbildung 4.6 – Die Messenger-Mission. Messengers Flug zum Merkur. Bild aus [113].

Definition 4.4.2 (Gravity Assist) *Gravity Assist bezeichnet ein Manöver in der Raumfahrt, bei der ein Flugkörper dicht an einem anderen Körper mit größerer Masse vorbeifliegt. Durch den Gravitationseinfluss des größeren Körpers ändert der Flugkörper seine Flugrichtung und -geschwindigkeit. Die Abbildung 4.7 zeigt jeweils zwei Beispiele mit Planeten für ein solches Manöver.*

Dieser Benchmark hat 26 verschiedene kontinuierliche Variablen, welche in der Tabelle 4.3 definiert sind:

Tabelle 4.3 – Beschreibung der Optimierungsvariablen von der Messenger Mission

Variable	Beschreibung
1	Starttermin gemessen von 1. Januar 2000 (MJD2000) [8]
2	Hyperbolische Überschussgeschwindigkeit am Anfang (km/s) [86]
3	Komponente der hyperbolischen Überschussgeschwindigkeit
4	Komponente der hyperbolischen Überschussgeschwindigkeit
$5 \sim 10$	Zeitintervall (in Tagen) zwischen Ereignissen (z. B. Gravity Assist)
$11 \sim 16$	Bruchteil des Zeitintervalls nachdem eine DSM* stattfindet [112]
$17 \sim 21$	Radius von Gravity Assist Manöver
$22 \sim 26$	Winkel gemessen in der Ebene des Planeten B des Planetenanflugvektors [112]

*DSM steht für *Deep Space Manouvre*.

Das Messenger Problem hat keine Einschränkungen, ausgenommen der Unter- und Obergrenzen der 26 Variablen.

Alle Space Mission Probleme basieren auf dem Multiple Gravity Assist Problem [42]. Mathematisch gesehen handelt es sich um ein endliches globales Optimierungsproblem mit nichtlinearen Einschränkungen. Es kann verwendet werden, um die bestmögliche Trajektorie zu finden, die eine mit einem chemischen Antriebsmotor ausgestattete interplanetare Sonde benötigt, um von der Erde zu einem anderen Planeten oder Asteroiden zu gelangen.

Die implementierten Modelle haben kleinere Abweichungen von der Realität wie z.B.: Es wird davon ausgegangen, dass die Anziehungskräfte der Planeten während der Gravity Asisst Manöver konstant bleiben.

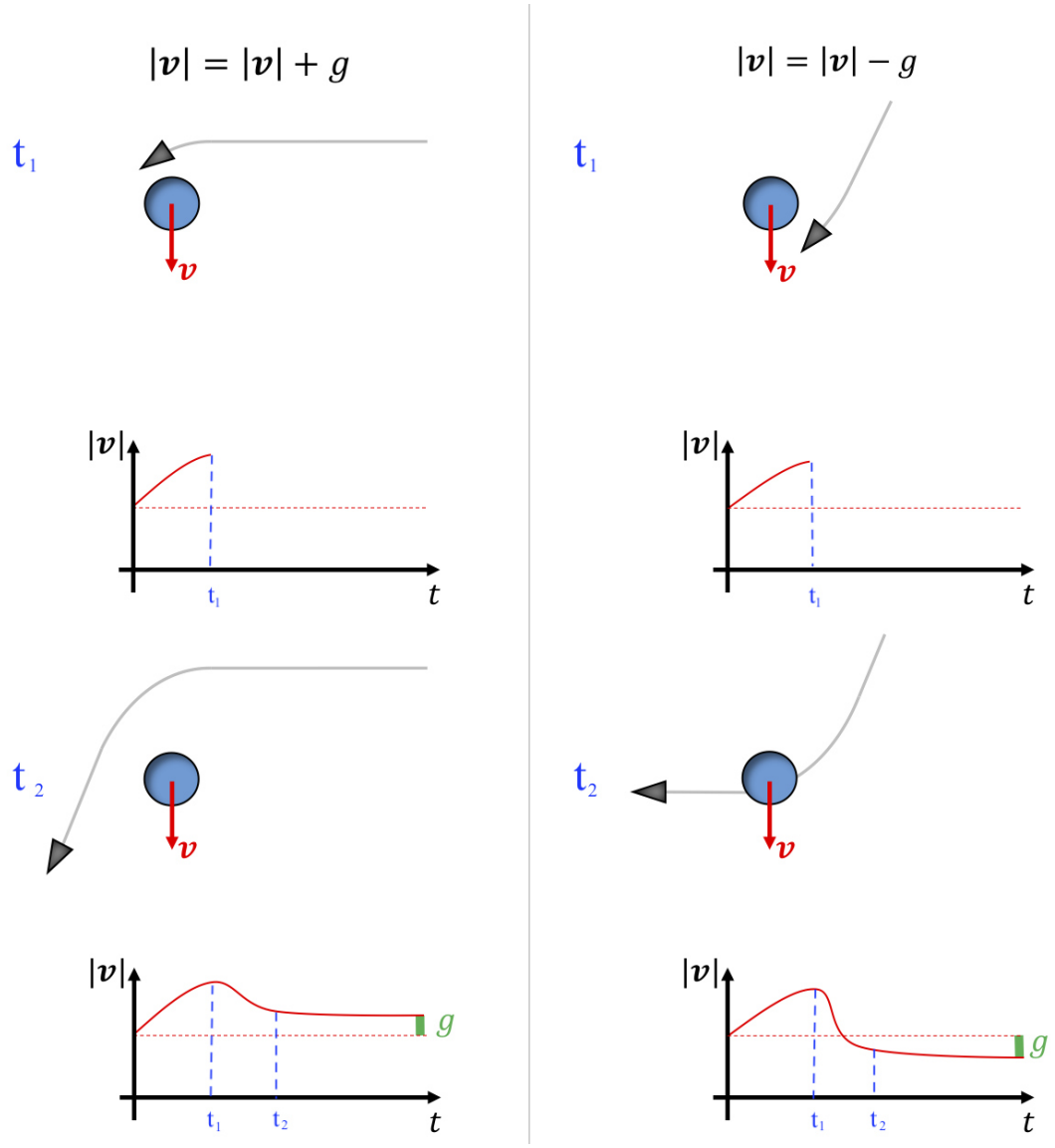


Abbildung 4.7 – Gravity Assist. Die linke Spalte zeigt den Ablauf des Gravitationseinflusses mit Geschwindigkeitszuwachs des Raumschiffes durch den mitziehenden Gravitationseinfluss des Himmelskörpers. Die rechte Spalte zeigt hingegen den Ablauf des Gravitationseinflusses mit Geschwindigkeitsabnahme des Raumschiffes durch den entgegen wirkenden Gravitationseinfluss des Himmelskörpers. In beiden Fällen entsteht eine Geschwindigkeitsänderung g und ein Ablenken der Richtung des Flugkörpers [122].

4.5 Rechnerarchitektur

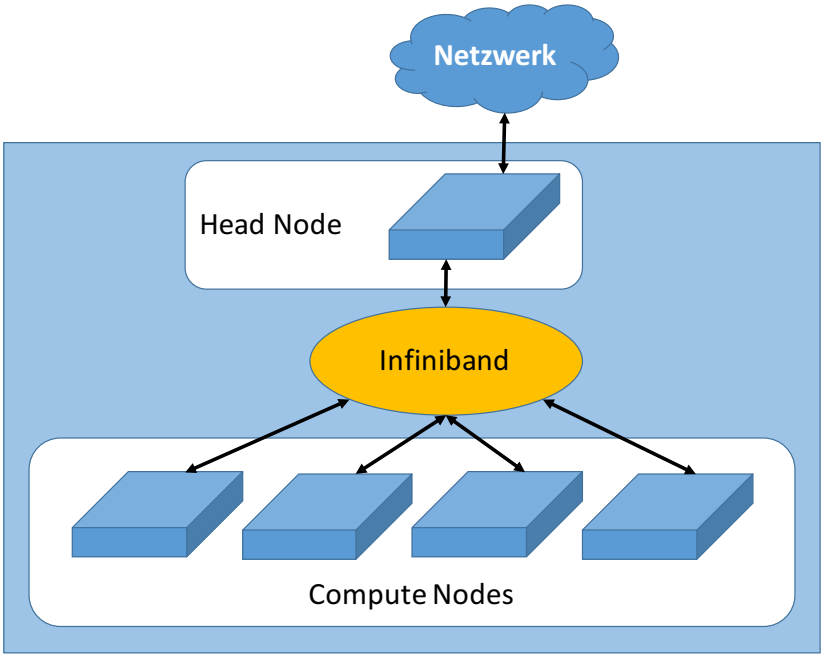


Abbildung 4.8 – High Performance Computing Cluster. Beispiel eines HPCC mit vier Rechnknoten.

Ein High Performance Computing Cluster (HPCC) ist ein Zusammenschluss aus mehreren Rechenknoten (s. Abb. 4.8), auf denen nebenläufig ein Problem evaluiert wird. Anwendung finden HPCCs insbesondere in der Forschung, da wir es mit sehr großen Datenmengen zu tun haben und durch HPCCs die Rechenzeit verkürzt werden kann. Die Rechenknoten sind miteinander verbunden und können kommunizieren. In der Regel gibt es hierbei dann einen Head Rechenknoten (eng. Node), der für die Verteilung der Aufgaben auf die einzelnen Rechenknoten zuständig ist. Jeder dieser Rechenknoten verfügt über einen eigenen Speicher sowie eigene CPUs und ist für sich genommen bereits ein eigener Server. Ein HPCC verwendet ein Wartenschlagen-System. Der Head Rechenknoten verwaltet die Ressourcen und verteilt Aufgaben, sobald ausreichend Ressourcen zur Verfügung stehen, was zu einer hohen Effizienz und weniger Leerlaufzeiten führt. Zum Zugriff aus Daten eines anderen Rechenknotens muss kommuniziert werden. Zwei Möglichkeiten dafür sind das Ethernet oder das InfiniBand.

4.5.1 SRA Cluster

Alle Evaluationen im folgenden Kapitel werden auf einem HPCC, den SRA Cluster, ausgeführt. Der SRA Cluster besteht aus 17 Rechenknoten und einem Head Rechenknoten. Ein Rechenknoten besteht aus zwei CPUs und jeder dieser CPUs besteht aus sechs Kernen. Die Kommunikation zwischen den Kernen innerhalb eines CPUs erfolgt durch einen Shared Memory Kanal. Die Kommunikation zwischen den verschiedenen Rechenknoten erfolgt über InfiniBand. Die Abbildung 4.9 zeigt die CPU Konfiguration eines Rechenknotens des SRA Clusters.

Die Hardware und Software Spezifikationen für den SRA Cluster sehen folgendermaßen aus:

CPU: Intel(R) Xeon(R) CPU X5650 @ 2,67GHz

RAM: 47,2 Gigabyte pro Rechenknoten

InfiniBand: Mellanox mhqh19b-xtr ConnectX-2 [105] mit 12 Lanes und einer Übertragungsgeschwindigkeit von 30 Gbit/S

Betriebssystem: CentOS Linux release 7.5.1804 (Core)

Kompiler: GCC Version 8.2.0 [26] und CMake Version 3.12.3 [81]

Cluster Manager Tool: TORQUE Version 5.1.0 [28]

Parallelisierung - Shared Memory: OpenMP Version 4.5 [117]

Parallelisierung - MPI: MVAPICH Version 2.3 [148]

Programmiersprache - C++: Version 2011 und Version 2014 [148]

Andere Softwarebibliotheken - Armadillo: Für die algebraischen Berechnungen wurde die C++ Bibliothek Armadillo mit der Version 9.200.4 benutzt [127].

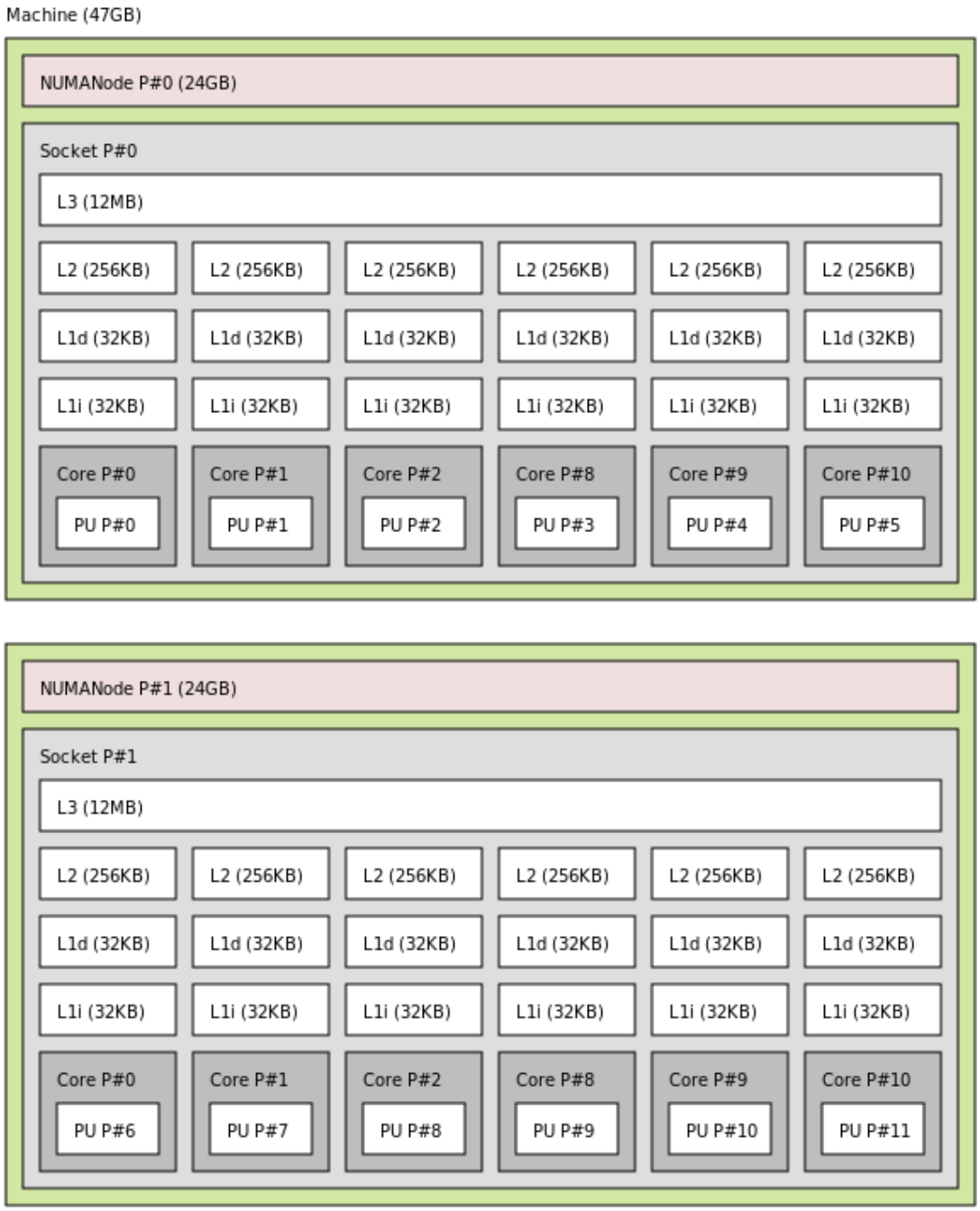


Abbildung 4.9 – SRA Cluster-Rechenknoten. Die CPU Konfiguration eines Rechenknotens in dem SRA HPCC.

4.6 Abgrenzung der Architektur

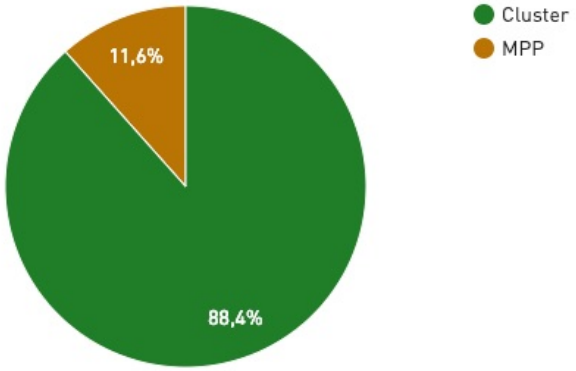


Abbildung 4.10 – Top 500 Rechner. Die Architektur der Top 500 Rechner aus [62].

Wie schon in dem vorherigen Kapitel, in dem eine Abgrenzung für die Algorithmen und Problemarten erbracht wurde, wird hier die Abgrenzung der Rechnerarchitektur geleistet. Wenn wir die Liste der Top 500 schnellsten Rechner weltweit anschauen (s. Abb. 4.10), dann sind 88,4% dieser Rechner mit einer Cluster Architektur ausgestattet. Deshalb wurde die PASS Implementierung genau für diese Art der Architektur entwickelt. Um OpenMP zu aktivieren wird zusätzlich ein Mehrkernprozessor mit Shared Memory benötigt. Ob es sich um eine Non-Uniform Memory Access (NUMA) oder Uniform-Memory-Access (UMA) Architektur sich handelt, ist für PASS irrelevant.

4.7 Zusammenfassung

Das vorliegende Kapitel hat einen Überblick über die Architektur von PASS gegeben. Es wurden die drei Module 1) Hardwareanalyse, 2) Parameterauswahl, 3) Schwarm-Suchalgorithmus vorgestellt und deren Ziel erklärt. Weiterhin wurden die Systemgrenzen der Methodik und der Softwareimplementierung diskutiert. Der PSO Algorithmus bildet die Basis der Methodik, deshalb wurde ein tieferer Blick in ihn gewährt. Darauffolgend wurden die zwei Gruppen der Evaluationsprobleme vorgestellt und näher betrachtet. Im Anschluss wurde die Rechnerarchitektur (SRA-

Cluster), in der die Evaluationen laufen, vorgestellt und die Abgrenzungen dieser Dissertation im Bezug zur Architektur erläutert.

Das nachfolgende Kapitel beschreibt die einzelnen Module im Detail und validiert sie mittels der BBOB Probleme.

5

PASS

Die Komponenten im Detail

> Was die Menschen am besten können, ist, neue Informationen so zu filtern, dass bestehende Auffassungen intakt bleiben.
>
> — Warren Buffet

Das nachfolgende Kapitel beschreibt die in Kapitel 4.1 vorgestellte Systemarchitektur im Detail. Dabei wird auf die einzelnen Module, deren Funktionsweise und deren Einordnung in das Framework eingegangen.

Als Erstes wird die Hardwareanalyse vorgestellt, als Nächstes folgt die Parameterauswahl und zum Schluss der Schwarm-Suchalgorithmus mit dem Island Model. Jedes dieser Module wird einzeln mit den BBOB Problemen evaluiert. Abschließend wird das Kapitel in Abschnitt 5.4 zusammengefasst.

5.1 Hardwareanalyse

5.1.1 Motivation

Traditionell wurde die Software für die serielle Berechnung geschrieben, indem das zu lösende Problem in eine Reihe von diskreten Instruktionen, welche sequentiell ausgeführt wurden, unterteilt wurde. Diese Instruktionen wurden von einem einzelnen Prozessor ausgeführt und es wurde nur je eine Instruktion gleichzeitig ausgeführt.

Die reale Welt ist aber massiv parallel. Viele komplexe, zusammenhängende Ereignisse geschehen zur gleichen Zeit. Hierzu gehören Ereignisse wie die Planetenbewegungen, die Klimaänderung, das Wetter, der Stau in der Hauptverkehrszeit oder das Bauen eines Autos. Das führt dazu, dass das parallele Rechnen viel besser geeignet ist, um diese Prozesse zu modellieren, zu simulieren und zu verstehen. Weiterhin sind die Vorteile des parallelen Rechnens enorm und einige davon sind:

1. **Zeit und/oder Geld sparen:** Das gleichzeitige Benutzen vieler Ressourcen führt zu einer Verkürzung der Zeit für die Lösung einer Aufgabe. Das wiederum könnte zu Kosteneinsparungen führen.

2. **Das Lösen komplexerer Probleme zu ermöglichen:** Heutzutage werden viele Probleme immer und immer größer, was mehr von den Ressourcen Zeit und Hardware fordert. Es gibt aber durchaus auch eine Menge Probleme, welche unmöglich durch nur einen einzelnen Rechner zu lösen sind. Eine Klasse dieser Probleme sind die *Grand Challenges* [64].

3. **Eine bessere Nutzung der zugrunde liegenden parallelen Hardware:** Die heutigen modernen Rechner sind parallele Rechner mit mehreren Prozessoren/Kernen. Die parallele Software ist speziell für eine parallele Hardware vorgesehen. In den meisten Fällen können serielle Programme einen modernen Rechner nicht voll auslasten und dadurch wird die Rechenleistung nur verschwendet.

Aus diesen Gründen wird das parallele Rechnen in vielen Gebieten wie in der Forschung, in der Entwicklung, in der Industrie und im Handel verwendet. Dieses sind die Gründe dafür, warum auch PASS eine parallele Ausführung ermöglicht.

Parallelisierung hängt immer mit dem Begriff *Speedup* zusammen. Der Speedup beschreibt mathematisch den Zusammenhang zwischen der seriellen und der parallelen Ausführungszeit eines Programmes. Speedup $\mathbf{S_p}$ wird wie folgt definiert [124]:

$$S_p = \frac{T_1}{T_p} \tag{5.1}$$

wobei $\mathbf{p}$ die Anzahl von Prozessoren ist und $\mathbf{T_1}$ und $\mathbf{T_p}$ entsprechend die Ausführungszeit auf einem Ein-Prozessor-System und auf einem Mehrprozessorsystem mit $\mathbf{p}$ Prozessoren sind.

Im Idealfall wäre der Speedup genau so groß wie die Anzahl der Prozessoren und wir hätten es mit einem Algorithmus zu tun, welcher sich zu 100% parallelisieren lässt. Das ist in der Praxis leider nicht erreichbar (siehe Amdahlsches Gesetz [6]), deshalb liegt der Wertebereich von $\mathbf{S_p}$ idealerweise zwischen $\mathbf{1}$ und $\mathbf{p}$. Die Abbildung 5.1 zeigt, wie eine ideale, lineare und reale Parallelisierungskurve abläuft. Ziel ist es, eine möglichst hohe Parallelisierbarkeit des Algorithmus zu erreichen.

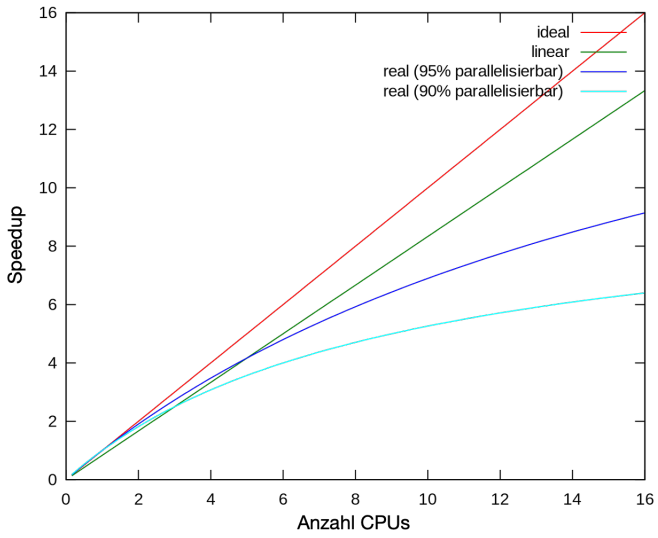


Abbildung 5.1 – Der Speedup einer parallel bearbeiteten Berechnung auf bis zu 16 CPUs [163].

Der Speedup trifft eine Aussage über den Zusammenhang auf einem einzelnen Rechner und ein Vergleich mit anderen Rechnern ist nicht möglich. Um verschiedene Implementierungen eines parallelen Algorithmus zu vergleichen, wird meistens eine objektive Kenngröße, die parallele **Effizienz**, benutzt. Die parallele Effizienz $\mathbf{E_p}$ wird wie folgt definiert:

$$E_p = \frac{S_p}{p} \tag{5.2}$$

5.1.2 Ziel

Das Ziel der *Hardwareanalyse* ist es, eine Vorhersage über den Speedup zu treffen. Anhand dieser Vorhersage kann entschieden werden, ob eine Parallelisierung erfolgen muss oder ob der Overhead größer als der Speedup ist und somit eine Parallelisierung nicht sinnvoll wäre. PASS bildet durch die *Hardwareanalyse* ein Kostenmodel der Architektur des Rechners und nutzt das Model, um eine Empfehlung für die Parallelisierung zu geben. Letztendlich liegt die endgültige Entscheidung bei dem Nutzer.

5.1.3 Arbeitsweise

Die *Hardwareanalyse* findet automatisiert statt und wird nur einmal durchgeführt. Als Ergebnis wird ein Kostenmodell der vorliegenden Architektur erstellt. Eine erneute Ausführung der *Hardwareanalyse* ist nur notwendig, wenn die Rechnerarchitektur sich ändert. Der ganze Prozess wird von Bildschirmausgaben (s. Abb. 5.2) begleitet, damit der Nutzer einen detaillierten Überblick bekommt, um die Ergebnisse besser nachzuvollziehen. PASS braucht als Übergabe für die Hardwareanalyse nur das zu optimierende Problem.

Der Ablauf dieser Analyse sieht wie folgt aus:

Problemanalyse: Das übergebene Problem wird analysiert. Die Dimension, der Name des Problems und die Zeit für eine Evaluation werden ausgegeben.

Hardwareprüfung: Die Hardware wird darauf geprüft, ob eine Parallelisierung überhaupt möglich ist. Es wird die maximale Anzahl der möglichen Threads übermittelt. Bietet die Hardware nur einen einzigen Thread, dann ist eine Parallelisierung nicht möglich. Ist diese Anzahl größer, wird sie vermerkt und der nächste Schritt folgt.

Modell-Check: Es wird geprüft, ob für die Hardware schon ein Kostenmodell existiert. Wird eines gefunden, dann wird dieses benutzt, um die Vorhersage zu machen, sonst folgt der nächste Schritt.

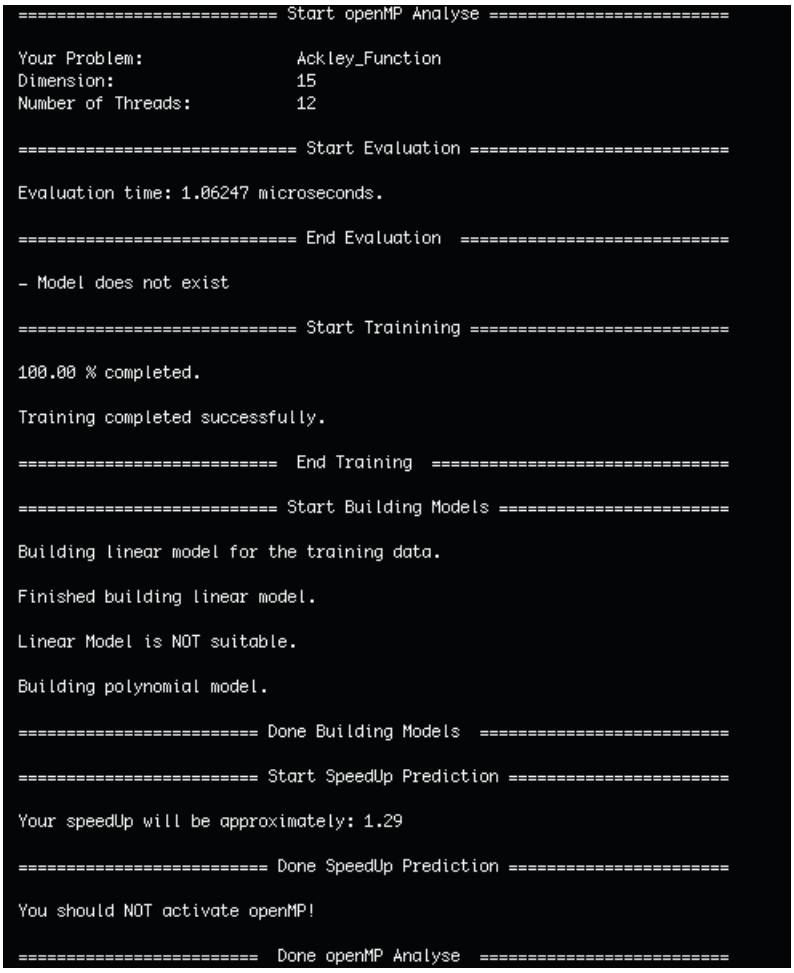


Abbildung 5.2 – Die Bildschirmausgabe während der Hardwareanalyse für die Ackley Funktion (F1).

Maschinelles Lernen: In diesem Punkt wird ein maschinelles Lernen angewendet. Die verwendete Methode ist die Regressionsanalyse [51]. Um das Modell anzutrainieren, müssen Daten erzeugt werden. Dadurch werden verschiedene Probleme mit verschiedenen Laufzeiten generiert und deren Evaluationszeit (für 200 Iterationen) für die serielle und parallele Ausführung gemessen. Durch die Messung wird der Speedup für die synthetisch erzeugten Probleme ermittelt. Je mehr Trainingspunkte gewählt werden, desto genauer ist das Modell. Das wiederum führt zu einer längeren Laufzeit der Hardwareanalyse.

Regressionsanalyse: Durch die Regressionsanalyse wird das Modell gebildet. Durch viele Tests hat sich ergeben, dass eine polynomiale Regression des dritten

Grades am besten geeignet ist. Die Abbildung 5.3 zeigt das maschinelle Lernen mit ca. 120 Trainingsdaten und das Ergebnis der Regression in dem SRA-Cluster (s. Abs. 4.5.1).

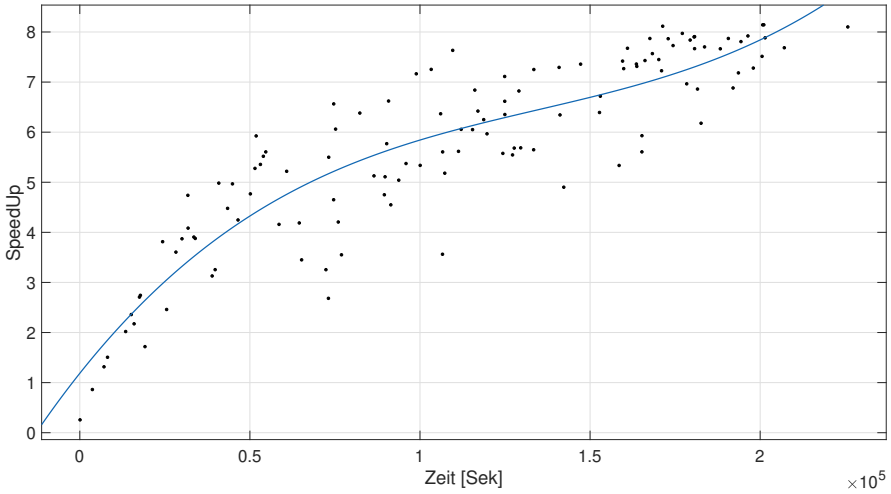


Abbildung 5.3 – Die Trainingsdaten und die resultierende Kurve des Modells aus der Regressionsanalyse auf dem SRA-Cluster mit 12 Rechenkerne.

Vorhersage: Anhand des gebildeten Modells wird eine Vorhersage über den Speedup getroffen und ausgegeben. Anhand der Effizienz wird dann eine Empfehlung für die Parallelisierung erteilt. Haben wir eine Effizienz niedriger als 10%, dann wird eine Parallelisierung nicht empfohlen. Liegt die Effizienz über 50%, wird eine Parallelisierung empfohlen und falls die Effizienz dazwischen liegt, soll der Nutzer anhand seiner Anforderungen und des Rechnersystems selber entscheiden, ob er eine Parallelisierung möchte.

Modellspeicherung: Das Modell wird dann in einer Datei für eine mögliche spätere Nutzung gespeichert.

5.1.4 Paralleler Particle Swarm Optimization Algorithmus

Durch die Hardwareanalyse wurde ermittelt, ob eine Parallelisierung möglich und zu empfehlen ist. Ist eine Parallelisierung nicht möglich, dann funktioniert PASS mit dem in Abschnitt 4.3 vorgestellten seriellen Algorithmus. Wird eine Parallelisierung angestrebt, so muss eine parallele Anpassung des Algorithmus folgen.

Wie schon im Kapitel 2.1.3 erwähnt, besteht die *online* Optimierung aus zwei Mechanismen, welche verantwortlich für die Gesamtzeit der Optimierung sind. Die Zeit des Evaluationsmechanismus $\mathbf{t_e}$ gibt die benötigte Zeit für die Evaluation eines Parameters aus dem Suchraum und die Zeit des Optimierungsmechanismus $\mathbf{t_o}$ gibt die Zeit für die Auswahl einer neuen Testlösung an. In der realen Welt ist $\mathbf{t_e}$ deutlich größer als $\mathbf{t_o}$. Weiterhin ist $\mathbf{t_e}$ immer von der Problemart abhängig und variiert von Fall zu Fall. $\mathbf{t_o}$ selbst ist immer konstant und in der Regel sehr klein. Das führt dazu, dass eine Parallelisierung des Optimierungsmechanismus mit einem Speedup $\mathbf{S_p < 1}$ verbunden ist, da der Overhead größer als die Ausführungszeit ist. Daher findet hier keine Parallelisierung statt. Nur der Evaluationsmechanismus kommt für eine Parallelisierung in Frage. Um die Entscheidung über die Parallelisierung zu treffen, muss ein Blick auf den SPSO2011 geworfen werden. Der SPSO2011 arbeitet asynchron und eine Umwandlung von asynchron zu synchron ist nötig, um die Parallelisierung zu ermöglichen. Die Algorithmen 5.1 und 5.2 zeigen, wie beide Verfahren funktionieren.

Die genauere Implementierung erfolgt über OpenMP. Durch einen Partikel pro Rechenkern wird die parallele Ausführung der Evaluationen ermöglicht. Mit dieser Änderung haben wir den SPSO2011 in seiner Arbeitsweise geändert. Deshalb soll geprüft werden, ob der neue parallele Algorithmus die gleiche Performance, bezüglich der Konvergenz, des „alten“ Algorithmus aufweist. Es gibt in der Fachliteratur einige Untersuchungen dieser Änderungen, aber sie beziehen sich auf die Vorgänger (z. B. SPSO2006 und SPSO2007) des SPSO2011 [25]. Aus diesen Gründen findet hier eine Evaluation statt, um die Performance des synchronen und asynchronen Algorithmus zu vergleichen.

```
while not stopping condition do
  for all Particles p in Swarm do
    p.evaluate();
    p.update();
  end for
end while
```

Algorithmus 5.1 – Asynchron

```
while not stopping condition do
  for all Particles p in Swarm do
    p.evaluate();
  end for
  for all Particles p in Swarm do
    p.update();
  end for
end while
```

Algorithmus 5.2 – Synchron

5.1.5 Evaluationen

Für die Evaluation wird ein sogenannter **A/B-Test** verwendet. Ein **A/B-Test** ist eine Testmethode zur Bewertung und Vergleich zweier Algorithmen, bei der die Originalversion gegen eine leicht geänderte Version getestet wird. Das Evaluationszenario sieht wie folgt aus:

Probleme: Der BBOB Benchmark mit den in Kapitel 4.4.1 definierten Funktionen F1 bis F8 wird mit den Dimensionen 5, 10, 15, 20 und 50 evaluiert (vgl. Kap. 3).

Wiederholung: Die Experimente werden 50 Mal für jedes Testszenario wiederholt.

Abbruchkriterium: Das Problem wurde mit einer Genauigkeit von $1e^{-6}$ gelöst oder eine maximale Anzahl von $10.000 \times Dimension$ Evaluationen wurde erreicht.

Gespeicherte Werte: Bei der Lösung des Problems wird die Anzahl der Iterationen gespeichert, ansonsten (bei 0% Erfolg) der beste erreichte Fitnesswert.

PSO Konfiguration: Der Algorithmus wird mit den empfohlenen Parametern gestartet [25]. Wir haben eine Population von 40 Partikeln sowohl für den asynchronen als auch für den synchronen Fall.

Es sind 40 Fälle, welche evaluiert werden. Mit der synchronen Version des Algorithmus wurde das Problem in **57%** der Fälle mindestens ein Mal gelöst und

bei der asynchronen Version des Algorithmus wurde das Problem in **65%** der Fälle mindestens ein Mal gelöst. Die Tabellen 5.1 und 5.2 zeigen das Minimum, Maximum und den Median der Iterationen, welche gebraucht wurden, bis das Problem gelöst wurde. Konnte das Problem mindestens ein Mal gelöst werden, dann werden nur diese Werte betrachtet. Wurde das Minimum nicht gefunden, dann werden die Fitnesswerte – siehe F5 in beide Tabellen – betrachtet, welche bis zum Erreichen der maximalen Anzahl von Iterationen erreicht wurden. Die Werte für die anderen Dimensionen sehen ähnlich aus.

	Minimum	Maximum	Median	St. Abw.	Erfolg
F1	205	228	218	5,379	100%
F2	90	111	101	4,374	100%
F3	613	1.185	722	262,934	80%
F4	228	1.249	496	258,942	42%
F5	6,3069 e-04	4,0943 e-03	2,2538 e-03	7,7955 e-04	0%
F6	222	1.014	359	253,75	16%
F7	64	235	82	34,770	94%
F8	53	803	92	115,507	100%

Tabelle 5.1 – Die Evaluationsdaten der synchronen Version für die 5-te Dim.

	Minimum	Maximum	Median	St. Abw.	Erfolg
F1	211	229	219	4,685	100%
F2	92	108	102	3,862	100%
F3	1.059	1.059	1.059	0	2%
F4	237	1.206	565	276,219	46%
F5	1,1391 e-03	1,1961 e-02	3,4777 e-03	1,9607 e-03	0%
F6	198	1.171	312	266,183	24%
F7	56	534	78	72,227	90%
F8	51	825	83	195,117	100%

Tabelle 5.2 – Die Evaluationsdaten der asynchronen Version für die 5-te Dim.

5.1.6 Fazit: Hardwareanalyse

Die ausgeführten Evaluationen haben die gleichen Ergebnisse für den SPSO2011 gezeigt, wie die Untersuchungen der Vorgänger Versionen gezeigt haben (s. Abs. 5.1.4). Die asynchrone Version konvergiert etwas schneller als die synchrone Version des Algorithmus. Weiterhin muss gesagt werden, dass die asynchrone Version mehr Probleme gelöst hat als die synchrone Version. Diese Unterschiede sind mit der Natur des Algorithmus verbunden. Durch die Zufallswerte sind solche Schwankungen ganz normal.

Die Evaluationen haben gezeigt, dass beide Versionen des Algorithmus benutzt werden können. Durch die Hardwareanalyse wird geprüft, ob das vorliegende Problem für eine Parallelisierung geeignet ist. Ist das der Fall, dann wird die parallele Version des Algorithmus verwendet, sonst die serielle Version.

Die Tests auf dem SRA-Cluster haben gezeigt, dass ab einer Evaluationszeit von **1,4** ms, eine Parallelisierung empfehlungswert ist. Diese Werte weichen selbstverständlich in anderen Rechnerarchitekturen ab.

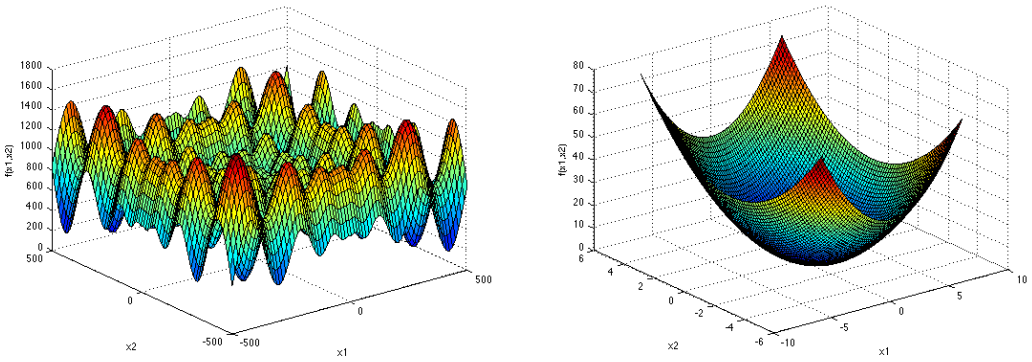
5.2 Parameterauswahl

5.2.1 Motivation

Die Performance des PSO Algorithmus ist, wie jeder andere metaheuristische Algorithmus, von der Auswahl der Parameter abhängig. Die optimalen Parameter hängen hauptsächlich von der Art des Problems ab [160]. Das Hauptproblem bei der Parameterauswahl ist ein Gleichgewicht zwischen der Exploration und der Exploitation zu finden. Die Exploration wird benötigt, um den Suchraum so gut wie möglich zu erforschen und die Exploitation wird benötigt, um sich auf einen bestimmten Bereich zu konzentrieren.

Die Anzahl der Partikel Pop, die Beschleunigungskonstante w, die lokale Anziehungskraft c_1, die globale Anziehungskraft c_2 und die Nachbarschaftstopologie np sind die Parameter des SPSO2011, welche konfiguriert werden müssen. Die richtigen Parameter hängen von der Fitnesslandschaft ab. Wenn es sich um eine flache Fitnesslandschaft handelt [39], dann ist die Wahrscheinlichkeit hoch, dass wenige lokale Minima zu finden sind, was dazu führt, dass weniger Exploration

benötigt wird. Das Gegenteil ist, wenn die Fitnesslandschaft hügelig ist [70]. Die Abbildung 5.4 zeigt Beispiele einer hügeligen und einer flachen Fitnesslandschaft. Welchen Einfluss die Nachbarschaftstopologie *np* hat, wurde schon in Kap. 4.3.2 erklärt. Die lokale Anziehungskraft c_1 führt dazu, dass die Partikel sich an ihrem lokal besten Wert orientieren, während die gloable Anziehungskraft c_2 die Nachbarschaft als Orientierung nimmt. Je größer c_1 und c_2 sind, desto stärker ist die Bewegung der Partikel vom eigenen besten Wert oder vom besten Wert der Nachbarschaft abhängig [22]. Die Beschleunigungskonstante *w* merkt sich die letzte Richtung des Flugs und führt dazu, dass keine großen Änderungen in der Bewegung der Partikel möglich sind [25]. Je größer der Wert der Beschleunigungskonstante ist, desto explorativer der Algorithmus.



(a) Beispiel einer hügeligen Fitnesslandschaft **(b)** Beispiel einer flachen Fitnesslandschaft

Abbildung 5.4 – Fitnesslandschaften zwei verschiedener Funktionen. Die hügelige Fitnesslandschaft gehört zur F6 Funktion und die flache Fitnesslandschaft gehört zur F2 Funktion.

In der Regel gibt es zwei Strategien, um die Parameter zu konfigurieren [160]. Die eine ist *online* und die andere ist *offline*. In der *offline* Parameterauswahl werden die Parameter vor der Initialisierung des Algorithmus gesetzt. In der *online* Parameterauswahl wird dieser Prozess während der Optimierung durchgeführt. Jede dieser Methoden hat ihre Vor- und Nachteile. Die *online* Parameterauswahl fängt mit den Standardparametern an und ändert sie bis die optimalen Parameter gefunden werden. Dieser Prozess dauert in der Regel länger als die *offline* Parameterauswahl, in der die optimalen Parameter schon gefunden wurden. Die in dieser Dissertation gewählte Methode ist die *offline* Parameterauswahl, weil dann eine schnelle Kon-

vergenz erreicht wird, wenn die Parameter vor der Optimierung gesetzt werden. Das Problem bei dieser Methode ist der große Wertebereich der Parameter. Wie schon in Kapitel 3 erwähnt, gibt es zwei Methoden, um die optimalen Parameter zu finden, welche aber sehr zeitintensiv sind.

5.2.2 Ziel

Das Ziel der Parameterauswahl ist das Finden einer guten Parameterkombination in einer kürzeren Zeit. Das kann erreicht werden, indem jeweils der Suchraum und die Dimension des Konfigurationsraums deutlich verkleinert werden. Um das zu erreichen, wird die existierende Fachliteratur untersucht. Maurice Clerc [25] und M. R. Bonyadi et. al. [14] haben sich mit der Eingrenzung der Wertebereiche für die Parameter beschäftigt. Durch verschiedene Tests konnten beide Gruppen die Wertebereiche der Parameter sehr gut einschränken (vgl. Tabelle 5.3). Wenn die Werte der Parameter außerhalb des Wertebereiches sind, dann kann das Verhalten des Algorithmus nicht erklärt werden [25]. Damit wurde der Wertebereich deutlich verkleinert.

Tabelle 5.3 – Wertebereich der Parameter des SPSO2011.

Parameter	Wertebereich
Pop	[5 ; 100]
w	[-1 ; 1]
$\mathbf{c_1}$	[0 ; 2,5]
$\mathbf{c_2}$	[0 ; 2,5]
np	[0 ; 1]

Nachdem der Wertebereich der Parameter deutlich verkleinert wurde, wird versucht, die Dimension der Suche zu verkleinern. Einige Wissenschaftler haben untersucht, welche die besten Parameterkombinationen für den PSO sind. Strößner hat diese in seiner Arbeit aufgelistet [141]. Diese Werte werden in der Tabelle 5.4 dargestellt.

Tabelle 5.4 – Liste der empfohlenen Parameter aus [141].

	w	c_1	c_2
Clerc & Kennedy	0,729	1,494	1,494
Trelea	0,6	1,7	1,7
Carlisle & Dozier	0,729	2,041	0,975
Jiang & Luo & Yang	0,715	1,7	1,7

Das Ziel ist es, einen Algorithmus zu haben, welcher weder die Exploration noch die Exploitation bevorzugt. Mit anderen Worten wird ein Gleichgewicht der beiden Verfahren angestrebt und das führt dazu, dass die Parameter c_1 und c_2 gleich sein müssten. Wird ein Blick auf die Tabelle 5.4 geworfen, ist einfach zu erkennen, dass in den meisten Fällen beide Parameter gleich sind. Darauffolgend wird die Hypothese gestellt, dass im Mittel die besten Ergebnisse erzielt werden, wenn die Parameter c_1 und c_2 gleich sind. Ist das der Fall, so kann der Suchraum um eine Dimension reduziert werden und statt fünf Parametern müssen nur vier untersucht werden. Dafür wird ein Experiment gestartet. Das Verfahren der Hauptkomponentenanalyse (s. Def. 5.2.1) wird benutzt, um festzustellen, ob eine Verkleinerung der Dimension möglich ist.

Definition 5.2.1 (Hauptkomponentenanalyse) *Die Hauptkomponentenanalyse wird benutzt um umfangreiche Datensätze zu vereinfachen, indem eine Vielzahl statistischer Variablen durch eine geringere Zahl möglichst aussagekräftiger Linearkombinationen (Hauptkomponenten) genähert wird [63].*

Durch die vielen ausgeführten Experimente und die gesammelten Daten wurde die Hauptkomponentenanalyse angewendet und es wurde festgestellt, dass die Variablen c_1 und c_2 für die betrachteten Testfälle nicht miteinander korrelieren und somit eine Reduzierung möglich ist.

Mit Hilfe der Literatursuche konnte der Suchraum deutlich verkleinert werden und die Dimension der Variablen wurde anhand der Hauptkomponentenanalyse von fünf auf vier reduziert. Im folgenden Abschnitt wird ein Algorithmus vorgestellt, welcher die Suche nach möglichst guten Parametern in kurzer Zeit ermöglicht.

5.2.3 Arbeitsweise

Die *Parameterauswahl* findet automatisiert statt und wird nur einmal durchgeführt. Als Ergebnis wird eine Datei mit den problemspezifischen besten Parametern erstellt.

Eine erneute Ausführung der *Parameterauswahl* ist nur notwendig, wenn sich das zu lösende Problem ändert. Der ganze Prozess wird mit Bildschirmausgaben (s. Abb. 5.5) begleitet, damit der Nutzer einen detaillierten Überblick bekommt, um die Ergebnisse besser nachvollziehen zu können. PASS braucht als Übergabe für die Hardwareanalyse nur das zu optimierende Problem.

Der Ablauf dieser Parameterauswahl sieht wie folgt aus:

Problemanalyse: Das übergebene Problem wird analysiert. Die Dimension, der Name des Problems und der verwendete Algorithmus werden ausgegeben.

Datei-Check: Es wird geprüft, ob für das vorliegende Problem schon eine Datei existiert. Wird eine gefunden, dann wird diese benutzt, sonst folgt der nächste Schritt.

Standardparameter: Der Algorithmus wird mit den Standardparametern gestartet. PASS benutzt den SPSO2011 als Algorithmus und die empfohlenen Standardparameter kommen aus [25].

Evaluation mit den Standardparametern: Das übergebene Problem wird mit dem Algorithmus evaluiert und die Ergebnisse werden für das spätere Vergleichen gespeichert. Die Anzahl der Evaluationen lässt sich steuern. Je höher diese ist, desto besser sind die Ergebnisse. Eine hohe Anzahl führt aber auch dazu, dass dieser Prozess länger dauert. Der Standardwert ist auf 50 gesetzt, kann aber nach Verlangen geändert werden.

Zufällige binäre Suche: Für alle Parameter des SPSO2011 wird eine zufällige binäre Suche ausgeführt. Die Funktionsweise des Algorithmus wird durch den Pseudocode 5.1 geschildert. Die Parameter werden nacheinander getestet. Die Tiefe der Suche kann durch einen Parameter (Granularität) bestimmt werden. Je größer dieser Parameter ist, desto genauer wird untersucht und desto länger dauert die Suche. Der Standardwert der Granularität ist auf 6 gesetzt. Die zufällige binäre Suche hat in den getesteten Fällen ca. 20 Minuten gedauert.

Standardparameter ersetzen: Findet die binäre Suche einen Parameter, welcher bessere Ergebnisse als der Standardparameter liefert, wird dieser mit dem gefundenen Parameter ersetzt. Falls nicht, bleibt der Standardparameter stehen. Diese Prüfung findet für alle Parameter statt.

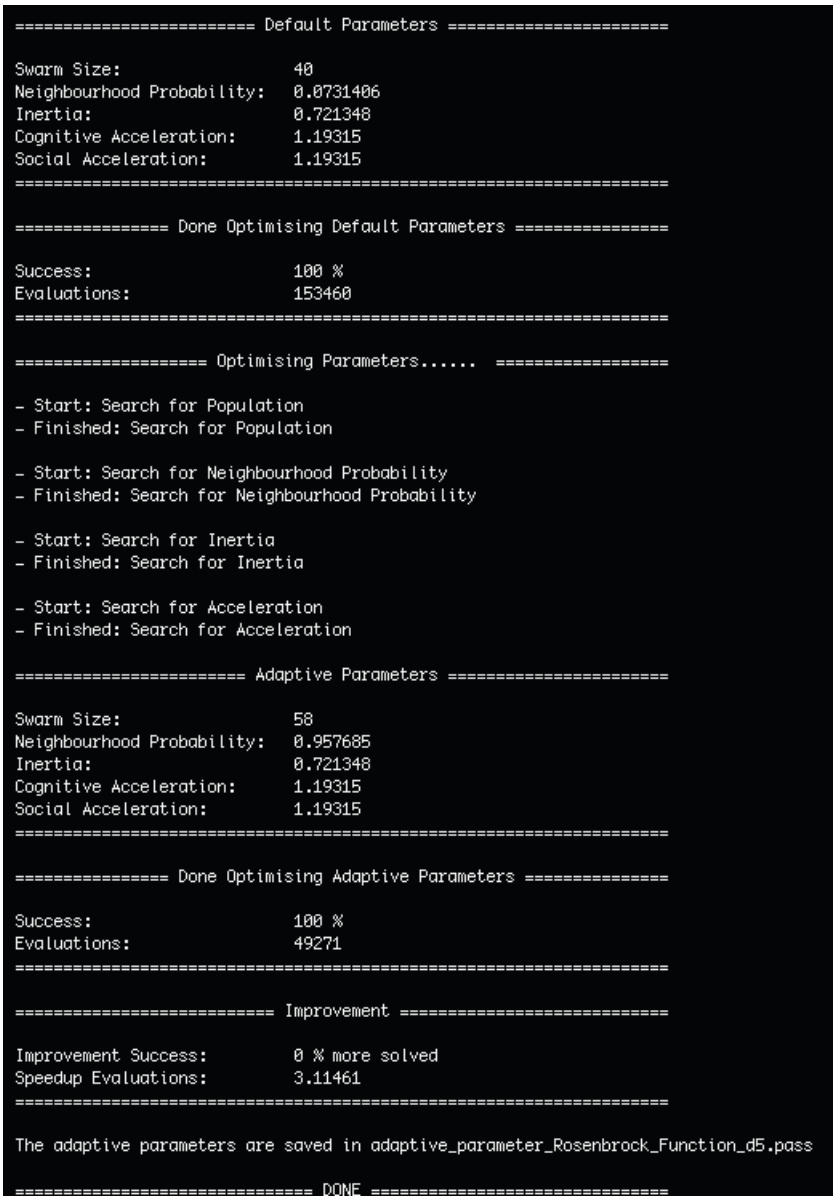


Abbildung 5.5 – Die Bildschirmausgabe während der Hardwareanalyse für die Rosenbrock Funktion (F5). Durch die Parameterauswahl könnte die Anzahl der Evaluationen um mehr als das Dreifache reduziert werden.

Ausgabe und Speicherung der besten gefundenen Parameter: Abschließend werden die besten gefundenen Parameter ausgegeben und in einer Datei für eine spätere Nutzung gespeichert. Zum Schluss wird auch ausgegeben, wie sehr sich die Ergebnisse im Vergleich zu dem Standardparameter verbessert haben.

```
    Eingabe low: Die untere Grenze des Wertebereichs des Parameters
    Eingabe high: Die obere Grenze des Wertebereichs des Parameters
    Eingabe granularity: Wie genau soll der Parameter sein
    Eingabe repeat: Die Wiederholungsanzahl der Optimierung
    Ausgabe parameter: Der untersuchte Parameter

1   BEGIN
2   for i = 1 to granularity do
3     mid = (high-low)/2 + low // bestimme die Mitte
4     left = random(low, mid) // wähle einen Wert aus der linken Seite
5     right = random(mid, high) // wähle einen Wert aus der rechten Seite
6
7     parameter = left  // setze den untersuchten Parameter auf den Wert left
8     for j = 1 to repeat do
9       score_left = evaluate-left()
10    end
11
12    parameter = right // setze den untersuchten Parameter auf den Wert right
13    for j = 1 to repeat do
14      score_right = evaluate-right()
15    end
16
17    if score_left < score_right do // vergleiche beide Seiten und ändere die Werte
18      high = mid
19      parameter = left
20    else
21      low = mid
22      parameter = right
23    end
24  end
25
26  parameter = compare(parameter, defaul_value) // vergleiche den besten gefundenen ↘
        Wert mit dem Standardwert und ersetze ihn ggf.
```

Pseudocode 5.1 – Zufällige binäre Suche

5.2.4 Evaluationen

Die Evaluationen sollen dazu dienen, die Performancesteigerung des SPSO2011 Algorithmus durch die Parameterauswahl zu zeigen. Verglichen wird der SPSO2011 mit und ohne Parameterauswahl.

Das Evaluationszenario sieht wie folgt aus:

Probleme: Der BBOB Benchmark mit den in Kapitel 4.4.1 definierten Funktionen F1 bis F8 wird mit den Dimensionen 5, 10, 15, 20 und 50 evaluiert (vgl. Kap. 3).

Wiederholung: Die Experimente werden 50 Mal für jedes Testszenario wiederholt.

Abbruchkriterium: Das Problem wurde mit einer Genauigkeit von $1e^{-6}$ gelöst oder eine maximale Anzahl von 2.000.000 Evaluationen wurde erreicht.

Gespeicherte Werte: Der Median der Anzahl der Evaluationen und die Erfolgsquote werden gespeichert und für den Vergleich benutzt.

PSO Konfiguration: Der Algorithmus wird mit dem empfohlenen Parameter gestartet [25].

Es werde 40 Fälle evaluiert. Mit der synchronen Version des Algorithmus wurde das Problem in **65%** der Fälle mindestens ein Mal gelöst und bei der asynchronen Version des Algorithmus wurde das Problem in **88,8%** der Fälle mindestens ein Mal gelöst. Die Tabellen 5.5 und 5.6 geben an, wie viel Prozent der Fälle gelöst wurden (links) und wie viele Evaluationen im Median gebraucht wurden (rechts). Je höher die Prozentanzahl und die Anzahl der Iterationen ist, desto besser. Ist das Problem sowohl ohne Parameterauswahl als auch mit Parameterauswahl nicht gelöst, dann wird der beste erreichbare Fitnesswert angegeben. Diese Fälle sind in den Tabellen grün markiert. Je kleiner dieser Wert ist, desto besser ist das Ergebnis.

	Dim. 5	Dim. 10	Dim. 15	Dim. 20	Dim. 50
F1	100% / 6.860	100% / 9.680	70% / 13.320	40% / 19.700	0%
F2	100% / 3.060	100% / 4.520	100% / 6.440	100% / 8.240	100% / 27.360
F3	10% / 10.073	20% / 10.600	20% / 12.420	0%	50% / 59.240
F4	40% / 28.880	0%	0%	14,4269	0%
F5	100% / 147.580	100% / 702.020	100% / 1,9271 e+06	0%	0%
F6	10% / 38.280	1075,88	1569,44	2744,92	7782,69
F7	90% / 3.200	10% / 4.360	10% / 5.240	0%	0%
F8	100% / 1.560	100% / 2.120	100% / 1.800	100% / 2.320	100% / 3.180

Tabelle 5.5 – Die Evaluationsdaten ohne die Parameterauswahl.

	Dim. 5	Dim. 10	Dim. 15	Dim. 20	Dim. 50
F1	100% / 5.320	100% / 9.660	100% / 16.013	100% / 24.202	80% / 79.709
F2	100% / 1.736	100% / 2.772	100% / 4.080	100% / 4.740	100% / 14.746
F3	20% / 20.646	70% / 31.363	80% / 15.568	90% / 20.328	90% / 53.100
F4	40% / 6.540	40% / 68.973	20% / 436.986	2,0588	10% / 877.816
F5	100% / 47.970	100% / 459.792	100% / 342.100	100% / 522.720	70% / 1,7890 e+06
F6	80% / 38.760	563.34	1461,58	2320,25	7321,31
F7	100% / 3.840	40% / 4.284	30% / 4.800	10% / 6.400	10% / 21.880
F8	100% / 516	100% / 1.462	100% / 1.420	100% / 1.800	100% / 2.214

Tabelle 5.6 – Die Evaluationsdaten mit der Parameterauswahl.

5.2.5 Fazit: Parameterauswahl

Wie erwartet führt die Parameterauswahl dazu, dass wir bessere Ergebnisse erzielen, da der Algorithmus besser an das Problem angepasst wird. Durch die Parameterauswahl wurden Probleme gelöst, welche vorher nicht gelöst wurden (z. B. F4 mit der Dimension 10 und 15) bzw. es wurden mehr Probleme gelöst als vorher (z. B. F6 Dimension 5, vorher 10%, nachher 80%). In den Fällen, in denen beide Verfahren 100% der Fälle gelöst haben, hat die Version mit Parameterauswahl schneller konvergiert als die ohne (z. B. F1 mit der Dimension 5 und 10).

5.3 Island Model

5.3.1 Motivation

Einer der Hauptvorteile von PSO ist die Fähigkeit, gute Ergebnisse in kürzester Zeit zu erzielen. Gleichzeitig ist diese Fähigkeit auch ein Nachteil des Algorithmus. In einer Studie [76], in der untersucht wurde, wie gut der PSO im Vergleich zu evolutionären Algorithmen ist, wurde betrachtet, dass der PSO schneller als die anderen Algorithmen konvergiert, aber die Qualität der Lösung nach einer bestimmten Anzahl von Evaluationen nicht besser wurde. Der Grund dafür ist, dass die Partikel in einem lokalen Minimum stagnieren. Das hat dazu geführt, dass die frühe Konvergenz untersucht wurde und verschiedene Methoden, um das Stagnieren zu vermeiden, vorgestellt wurden. Wie schon in Kapitel 3 erwähnt wurde, gibt es hier verschiedene Behandlungsweisen und diejenige, die sich die letzten Jahren als am effizientesten erwiesen hat, ist die *Island Model* Methode.

Das Ziel von *Island Model* ist die Vermeidung der frühen Konvergenz, in der die Diversität verschiedener Schwärme benutzt wird.

5.3.2 Arbeitsweise

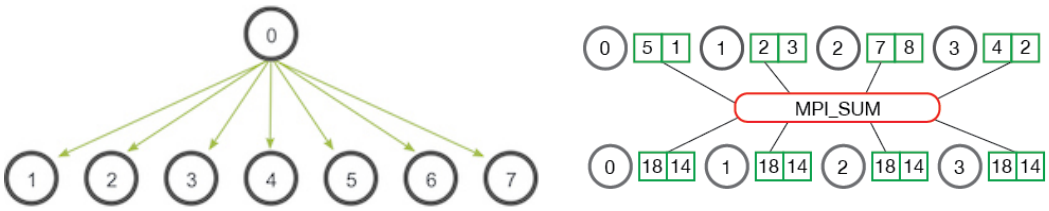
Das Island Model wurde erstmals in [97] vorgeschlagen. Dieses erste Modell wurde im Verlauf der Jahre immer wieder perfektioniert und weiterentwickelt. Mit dem Aufkommen der Parallelrechner gewann dieses Modell an Bedeutung. Ziel ist es, die Lösungsqualität zu verbessern und damit die Stagnation zu verzögern. Bei einer parallelen Implementierung eines Inselmodells führt jeder Rechenknoten einen PSO-Schwarm aus und Informationen werden in bestimmten Intervallen zwischen den Schwärmen ausgetauscht. Ein Multi-Schwarm kann durch ein Tripel definiert werden $\langle \mathscr{MP}, \mathscr{T}, \mathscr{C} \rangle$, in dem $\mathscr{MP} = \{MP_1, ..., MP_n\}$ die Anzahl der Schwärme, $\mathscr{T}$ die Topologie der Schwärme und $\mathscr{C}$ die Migrationsstrategie ist.

$\mathscr{C}$ selber ist durch vier Variablen definiert $\langle \alpha, \beta, \gamma, \delta \rangle$, wobei α die Häufigkeit des Austausches ist, β die Anzahl der Partikel zeigt, welche Partikel von einem Schwarm zum anderen migrieren, γ angibt, welche Partikel migriert werden, und δ anführt, welche Partikel ersetzt werden.

Für $\mathscr{T}$ wird eine maximale globale Kommunikation angestrebt (d.h. jeder Schwarm kommuniziert mit allen anderen). Nach jeder Iteration (α), wird der schlechteste Partikel jedes Schwarmes (δ) mit dem besten globalen Partikel (γ) ausgetauscht.

5.3.3 Implementierung

Die Implementierung des *Island Models* folgt über MPI. Die Architektur der Implementierung wurde schon in Kapitel 3.4 erläutert. Für das Senden der Nachrichten über Infiniband werden zwei MPI-Befehle benutzt: 1) *MPI_Allreduce* und 2) *MPI_Bcast*. Durch den *MPI_Allreduce* wird zuerst ermittelt, welcher Knoten den Partikel mit dem besten Fitnesswert besitzt und danach schickt dieser Knoten diesen Partikel durch *MPI_Bcast* an alle anderen Knoten. Diese ersetzen den Partikel mit dem schlechtesten Fitnesswert mit dem geschickten Partikel aus. Die Abbildung 5.6 zeigt eine grafische Darstellung dieser zwei Befehle.



(a) MPI_Bcast **(b)** MPI_Allreduce

Abbildung 5.6 – Die benutzten MPI-Befehle. Für den *MPI_Allreduce* Befehl wird die Operation *MPI_SUM*, welche die Summe der gelieferten Werte bildet, benutzt.[87]

5.3.4 Evaluationen

Die Evaluationen sollen dazu dienen, den Einfluss des Island Models auf den SPSO2011 Algorithmus zu untersuchen. Verglichen wird der SPSO2011 mit und ohne Island Model.

Das Evaluationszenario sieht wie folgt aus:

Probleme: Der BBOB Benchmark mit den in Kapitel 4.4.1 definierten Funktionen F1 bis F8 wird mit den Dimensionen 5, 10, 15, 20 und 50 evaluiert (vgl. Kap. 3).

Widerholung: Die Experimente werden 50 Mal für jedes Testszenario wiederholt.

Abbruchkriterium: Das Problem wurde mit einer Genauigkeit von $10e^{-6}$ gelöst oder eine maximale Anzahl von $10.000 \times Dimension$ Evaluationen wurde erreicht.

Gespeicherte Werte: Das Minimum, das Maximum, der Median, die Standardabweichung der Anzahl der Iterationen und die Erfolgsquote werden gespeichert und für den Vergleich benutzt.

PSO Konfiguration: Der Algorithmus wird mit den empfohlenen Parametern gestartet [25].

Island Model Konfiguration: Für α (Häufigkeit des Austausches) werden die Werte 0, 1, 2 und 3 getestet und für $\mathscr{MP}$ (die Anzahl der Schwärme bzw. Nodes) werden die Werte 2, 6, 8, 10, 12, 14 und 16 getestet.

Es sind insgesamt **2.560** Kombinationen, welche getestet wurden. Es hat sich gezeigt, dass $\alpha = 0$ die besten Ergebnisse liefert. D.h. nach jeder Iteration findet der Austausch statt. Ein $\alpha > 0$ liefert schlechte Ergebnisse, weil die neuen Informationen zu veraltet sind und kaum Einfluss auf die Ergebnisse haben. Wenn die Schwarmgröße aus durchschnittlich 40 Partikeln besteht, dann ist ein Austausch nach 40 Evaluationen bzw. eine Iteration das Beste. Weiterhin wurde auch gezeigt, dass für eine steigende Anzahl an Schwärmen die Wahrscheinlichkeit umso höher ist, dass die lokalen Minima vermieden werden.

Die Tabelle 5.7 gibt einen Überblick über den Einfluss der Häufigkeit des Austausches. Es wurden exemplarisch einige Fälle aus den 2.560 Kombinationen ausgesucht. Die Anzahl der benutzten Schwärme war zwei. Tabelle 5.8 gibt einen Überblick über den Einfluss der Anzahl der Schwärme. Auch hier wurden exemplarisch einige Fälle ausgesucht. Für alle Fälle ist $\alpha = 0$.

In beiden Tabellen geben die Zahlen in Klammern neben den jeweiligen Problemnamen die Dimension des Problems an. Die Zahlen in den Tabellen geben an, wie viel Prozent der Fälle gelöst wurden (links) und wie viele Evaluationen im

Median gebraucht wurden (rechts). Je höher die Prozentzahl und je niedriger die Anzahl der Evaluationen ist, desto besser sind die Ergebnisse. Die besten Ergebnisse sind grün markiert.

	$\alpha = 0$	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
F1 (Dim. 10)	100% / 8.480	100% / 9.480	100% / 1.020	100% / 1.021
F2 (Dim. 50)	100% / 19.320	100% / 23.400	100% / 24.540	100% / 25.540
F3 (Dim. 5)	40% / 11.640	10% / 18.800	18% / 17.800	16% / 18.940
F4 (Dim. 5)	98% / 17.120	92% / 23.820	88% / 44.100	74% / 40.840
F5 (Dim. 10)	98% / 31.240	98% / 32.360	88% / 32.360	18% / 38.640
F6 (Dim. 5)	52% / 16.800	38% / 17.200	48% / 33.400	26% / 18.800
F7 (Dim. 50)	8% / 14.660	4% / 18.180	0%	6% / 21.720
F8 (Dim. 50)	100% / 2.100	100% / 2.700	100% / 2.640	100% / 2.580

Tabelle 5.7 – Die Evaluationsdaten für das Island Model mit verschiedenen Kommunikationsstrategien.

	$\mathscr{MP} = 2$	$\mathscr{MP} = 6$	$\mathscr{MP} = 10$	$\mathscr{MP} = 14$	$\mathscr{MP} = 16$
F1 (Dim. 10)	100% / 8.480	100% / 7.880	100% / 7.720	100% / 7.680	100% / 7.600
F2 (Dim. 20)	100% / 6.700	100% / 5.400	100% / 5.120	100% / 4.960	100% / 4.920
F3 (Dim. 15)	38% / 9.600	76% / 7.980	76% / 7.280	78% / 7.360	80% / 7.320
F4 (Dim. 5)	98% / 17.120	100% / 9.360	100% / 8.420	100% / 8.400	100% / 7.720
F5 (Dim. 10)	98% / 31.240	100% / 18.540	100% / 19.400	100% / 19.640	100% / 18.010
F6 (Dim. 5)	38% / 17.520	84% / 16.200	96% / 13.000	98% / 10.640	98% / 9.440
F7 (Dim. 50)	8% / 14.660	32% / 10.220	62% / 8.920	58% / 8.280	70% / 7.960
F8 (Dim. 20)	100% / 1.960	100% / 1.580	100% / 1.420	100% / 1.300	100% / 1.300

Tabelle 5.8 – Die Evaluationsdaten für das Island Model mit verschiedenen Schwarmgrößen.

Je mehr Schwärme benutzt werden, desto größer ist der Overhead. Das ist ein Problem, welches entsteht, wenn wir das Optimum sowohl aus Dauer als auch aus Genauigkeit bestimmen wollen. Teil dieser Arbeit ist nur die Genauigkeit, aber die Zusammenhänge werden dennoch kurz erläutert.

Dauer vs. Genauigkeit

Wie anhand der Evaluationen gesehen werden kann, ist es umso besser, je öfter kommuniziert wird. Die MPI Kommunikation ist aber mit einem Overhead verbunden. Teil dieser Arbeit ist jedoch nur die Genauigkeit der Ergebnisse und nicht der Zeitaufwand, welcher durch die MPI Befehle erzeugt wird. Nichtsdestotrotz werden einige Aspekte des zeitlichen Aufwands angesprochen. Es gibt verschiedene Methoden, wie man den zeitlichen Aufwand eines MPI Befehls errechnen kann, und eine davon ist die Methode von R. Thakur et. al. [147]. Der zeitliche Aufwand einer Kommunikation lässt sich in etwa durch *MPI_Allreduce* anhand der Formel 5.3 beschreiben. Hier steht **D** für die Anzahl von Dimensionen und **N** für die Anzahl von Nodes, $\mathbf{t_{init}}$ repräsentiert die Zeit, die zur Initialisierung benötigt wird, und $\mathbf{t_{send}}$ die Dauer einer Übertragung eines Elementes.

$$T_{Allreduce} = (N-1) \cdot t_{init} + \frac{N-1}{N}(D+1) \cdot t_{send} \tag{5.3}$$

Wenn man davon ausgeht, dass die Initialisierungszeit bei demselben Rechner gleich ist und die Dimension des Problems sich nicht ändert, dann ist die Gesamtzeit von der Anzahl der Nodes und der Größe der Nachricht abhängig. Das ist von MPI-Befehl zu MPI-Befehl unterschiedlich. Die Abbildungen 5.7 und 5.8 stellen diese Zusammenhänge für *MPI_Allreduce* und *MPI_BCast* auf dem SRA-Cluster dar. Wie den Grafiken entnommen werden kann, ist der Overhead bzw. die Gesamtzeit der Übertragung umso größer, je mehr Nodes vorhanden sind und je größer die Nachricht ist. In Abhängigkeit des Zieles einer Optimierung sollte entschieden werden, ob mehr Wert auf die Genauigkeit oder Dauer gelegt werden sollte. Um das Optimum finden zu können, sollte das Ganze in ein Optimierungsproblem umgewandelt werden, in dem zwei Variablen optimiert werden müssen (s. Paretofront in Abs. 2.1.2).

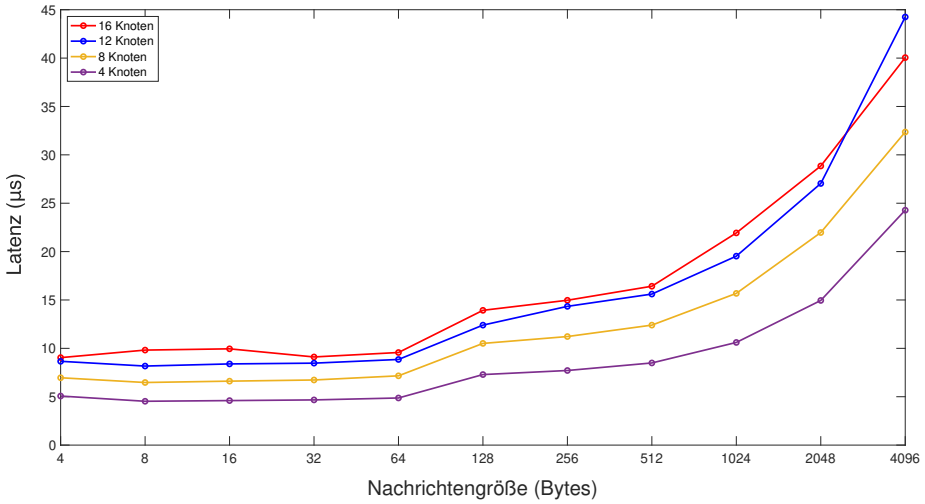


Abbildung 5.7 – Die Gesamtzeit von *MPI_Allreduce* in dem SRA-Cluster in Abhängigkeit von der Anzahl der Nodes und der Nachrichtengröße.

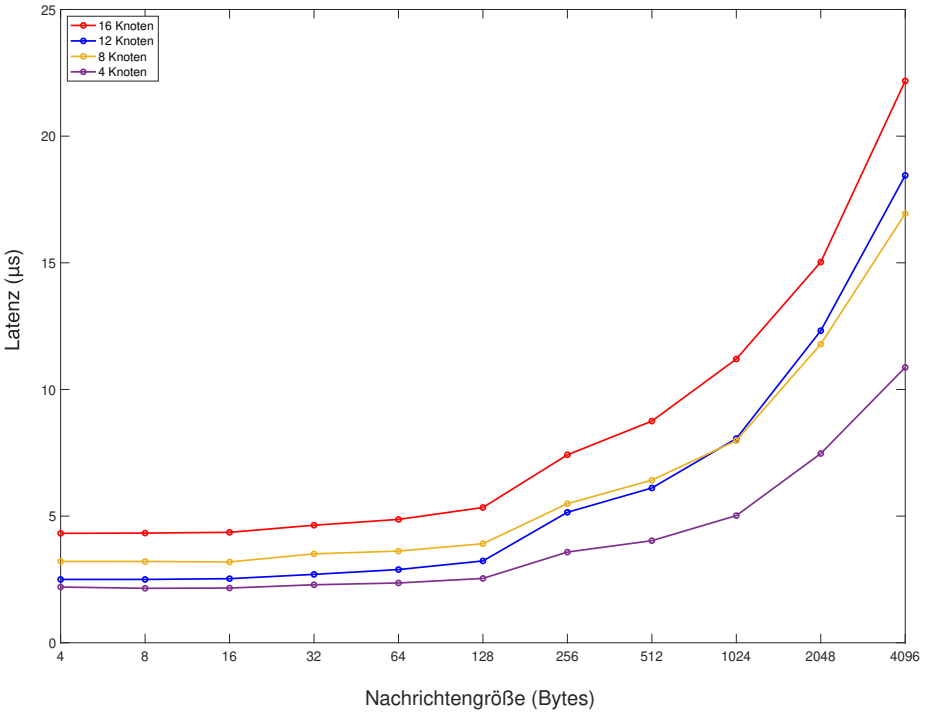


Abbildung 5.8 – Die Gesamtzeit von *MPI_BCast* in dem SRA-Cluster in Abhängigkeit von der Anzahl der Nodes und der Nachrichtengröße.

5.4 Zusammenfassung

In diesem Kapitel wurden die Einzelteile von PASS validiert. Die Hardwareanalyse wird nur für den Fall aktiviert, dass sie Vorteile bei der Geschwindigkeit bringt. Weiterhin wurde gezeigt, dass durch die parallele Änderung des Algorithmus keine Änderungen in der Konvergenz folgen. Das alles passiert automatisch und der Nutzer muss sich um nichts kümmern.

Als Zweites wurde die Parameterauswahl validiert und erwartungsgemäß sind die Ergebnisse nach der Parameterauswahl besser als zuvor ohne. Das ist einfach zu erklären, weil der Algorithmus durch eine genauere Anpassung der Parameter besser an den Eigenschaften der Fitnesslandschaft ausgerichtet ist. Der vorgestellte Algorithmus für die Parametersuche verkürzt deutlich die Zeit, welche die vorherigen Methoden gebraucht haben. Im verwendeten SRA-Cluster mit der aktuellen Konfiguration dauert eine Parametersuche in der Regel um die 20 bis 30 Minuten.

Zuletzt wurde das Island Model validiert. Genau wie oben auch waren die Ergebnisse hier besser als ohne das Island Model. Durch diese Methode wird eine frühe Konvergenz vermieden. Das Island Model wurde mittels MPI implementiert und die Zeitprobleme, welche durch MPI entstehen können, wurde erläutert und diskutiert.

Zusammenfassend bringen die drei implementierten Methoden Vorteile für die Genauigkeit des SPSO2011 Algorithmus. PASS hat diese drei Methoden implementiert und die Schritte für das Prüfen und Aktivieren der Methoden werden automatisch getroffen. Keine weiteren Fachkenntnisse des Algorithmus oder der verwendeten Architektur werden benötigt. Der Vorteil, den die Methoden bringen, wurde durch den BBOB Benchmark gezeigt.

Im nachfolgenden Kapitel wird PASS mit anderen Algorithmen aus dem Stand der Technik verglichen. Als Vergleich dienen die Space Mission Probleme aus Kap. 4.4.2. Im Unterschied zu diesem Kapitel, werden für die kommenden Evaluationen alle drei vorgestellten Methoden (Hardwareanalyse, Parameterauswahl und Island Model) in PASS aktiviert.

6

Evaluationen

Probleme aus der realen Welt

> Es ist nicht genug zu wissen, man muss es auch anwenden. Es ist nicht genug zu wollen, man muss es auch tun.
>
> JOHANN WOLFGANG VON GOETHE

Das nachfolgende Kapitel beschäftigt sich mit den Evaluationen der Space Mission Probleme (s. Kap. 4.4.2). Es wurden einige sehr bekannte und effiziente Algorithmen aus dem Stand der Technik ausgewählt und als Benchmark mit der PASS-Lösung verglichen.

Als Erstes wird in diesem Kapitel das Ziel der Evaluationen vorgestellt. Als Nächstes folgen die Algorithmenkonfiguration und das Szenario und zum Schluss die Auswertung der Evaluationen. Abschließend wird das Kapitel in Abschnitt 6.4 zusammengefasst.

6.1 Ziel der Evaluation

Das Ziel der Evaluation ist es zu zeigen, dass PASS im Durchschnitt bessere Ergebnisse liefert als andere Algorithmen, welche für eine bestimmte Domäne vorgesehen sind. Die Evaluationen sollten auch zeigen, dass durch das Zusammenspiel der drei in Kap. 5 vorgestellten Bestandteile von PASS (Hardwareanalyse, Parameterauswahl und Island Model) die Performance eines Algorithmus deutlich erhöht werden kann. In Kapitel 3 wurde dieses Ziel der vorliegenden Dissertation genau definiert.

6.2 Szenario und Algorithmenkonfiguration

Für die Evaluationen wurden einige Algorithmen, welche auf verschiedenen Paradigmen basieren, ausgewählt. Die Selektion der Algorithmen ist zwar nicht lückenlos, aber sie beinhaltet einige der bekanntesten Algorithmen, welche in den letzten Jahren gezeigt haben, dass sie gute Ergebnisse liefern. Eine genauere Beschreibung der Arbeitsweise der Algorithmen ist in Kap. 3 gegeben. Die ESA liefert die ausgesuchten Algorithmen und deren Implementierung durch ein Framework (PAGMO)[6] und dieses wird dann benutzt, um die Evaluationen durchzuführen. Die Ergebnisse dieser Evaluation werden mit den Ergebnissen, welche durch PASS geliefert werden, verglichen. Die Konfiguration der Algorithmen sieht wie folgt aus:

Simulated Annealing (SA): Es gibt verschiedene Implementierungen von SA. In dieser Arbeit wird die Version von Corana et. al [29] benutzt, welche auf einer adaptiven Nachbarschaft basiert. Die Hauptparameter des Algorithmus sind die Start- und Finaltemperatur (T_s, T_f). Diese Werte sind mit $T_s = 10$ und $T_f = 0.1$ initialisiert. Weitere Parameter, welche die Performance des Algorithmus beeinflussen, sind N_s und N_T, die respektive die Anzahl der Zyklen und die Anzahl der Suchintervalländerungen pro Temperaturschritt angeben (siehe Corana et. al [29]). Diese Werte sind mit $N_s = 20$ und $N_T = 1$ initialisiert.

Differential Evolution (DE): Die benutzte Version ist die von Storn und Price [139]. Dieser Algorithmus besitzt zwei Varianten, welche *rand/1/exp* und *rand/1/bin* genannt werden. Der Unterschied liegt nur darin, wie die Rekombination (eg. Crossover) vollzogen wird (binominal oder exponentiell). Die Parameter des

[6] https://esa.github.io/pagmo2/docs/algorithm_list.html

Algorithmus sind wie folgt gesetzt: $CR = 0.9$ und $F = 0.8$ und die benutzte Variante ist die, welche eine exponentielle Rekombination erlaubt.

Covariance Matrix Adaptation Evolution Strategy (CMA-ES): Dieser ist einer der bekanntesten Algorithmen in der Domäne der Black-box Probleme. Die benutzte Version ist die von Hansen et. al [68]. Die Parameterkonfiguration wird gesetzt, wie es im Paper empfohlen wird.

Artificial Bee Colony (ABC): Die benutzte Version ist die von Karaboga [83]. Die Parameter dieses Algorithmus sind die Populationsgröße SN und $limit$. Hat ein Agent nach *limit* Versuchen keinen Parameter mit einem besseren Fitnesswert gefunden, wechselt er von „beschäftigt“ auf „erkunden“. Beide Werte sind initialisiert als $SN = 20$ und $limit = 2 \times SN$.

Standard Particle Swarm Optimization 2011 (SPSO2011): Die benutzte Version ist die letzte, welche von Clerc et. al[159] vorgestellt wurde. Die Konfiguration der Parameter wurde so gesetzt, wie es im Paper von Clerc [25] empfohlen wird.

Parallel Adaptive Swarm Search (PASS): Für den PASS Algorithmus wurden alle drei Komponenten (s. Kap 5) aktiviert. Die Parameterkonfiguration erfolgt automatisch und wird vor dem Ausführen des Algorithmus durchgeführt. Das Island Model wird aktiviert und es werden 16 Nodes in dem SRA-Cluster aktiviert.

Nachdem die Konfiguration der Algorithmen vorgestellt wurde, wird auch das Szenario der Evaluationen definiert:

Probleme: Die Space Mission Probleme (s. Kap. 4.4.2 und 4.4.3) werden evaluiert.

Wiederholung: Die Experimente werden 1.000 Mal wiederholt.

Abbruchkriterium: Da es sich hier um echte Blackbox Probleme handelt, wird die Evaluation erst nach 500.000 Funktionsevaluationen unterbrochen.

Gespeicherte Werte: Das Minimum, das Maximum, der Median und die Standardabweichung der erreichten Fitnesswerte werden gespeichert und für den Vergleich benutzt.

Bedeutung Werte: Je kleiner die Fitnesswerte sind, desto besser ist das Ergebnis.

6.3 Evaluationen

In diesem Abschnitt werden die Evaluationen als Grafik vorgestellt und zusammengefasst. Für jedes einzelne Problem werden folgende Grafiken erzeugt:

Tabelle: In der Tabelle sind das Minimum, das Maximum, der Median und die Standardabweichung der gefundenen Fitnesswerte aufgelistet. Die Werte beziehen sich auf 1.000 Evaluationen. Der beste Medianwert aller Algorithmen ist grün markiert.

Histogram: Das Histogram stellt die Häufigkeitsverteilung der Evaluationen dar. Die X-Achse repräsentiert die erreichten Fitnesswerte und die Y-Achse repräsentiert die Häufigkeit des Auftretens dieser Werte. Die rote Linie zeigt die normale Wahrscheinlichkeitsverteilung, d.h. die Wahrscheinlichkeit, dass der Wert X bei einer Wiederholung eintritt.

Box-Plot: Der Box-Plot ist ein Diagramm, das zur grafischen Darstellung der Verteilung eines mindestens ordinalskalierten Merkmals verwendet wird [94]. Durch die Darstellung wird schneller übermittelt, in welchem Bereich die Daten liegen und wie die Verteilung dieser Daten ist. In jedem Kästchen zeigt die zentrale Markierung den Median an, die untere und obere Kante des Kästchens zeigt das 25. und 75. Perzentil an. Die Whisker (Antenne außerhalb des Kästchens) reichen bis zu den extremsten Datenpunkten, die nicht als Ausreißer betrachtet werden, und die Ausreißer werden einzeln mit dem Symbol "+"dargestellt. In der X-Achse werden die Algorithmen aufgelistet und die Y-Achse repräsentiert die erreichten Fitnesswerte.

Durch die Auswahl der Algorithmen wurden Exemplare aus beiden Oberklassen der statistischen Optimerungsmethoden und deren Unterklassen ausgewählt (s. Abb. 3.1). SA ist Repräsentant der trajektorienbasierten Methoden. DE und CMA-ES sind Repräsentanten der evolutionären Algorithmen, ABC und PSO sind Repräsentanten der schwarmbasierten Methoden. Diese Algorithmen repräsentieren eine Gruppe des

letzten Standes der Technik und werden allgemein in der Forschung als Benchmark für den Vergleich benutzt.

Der Median und die Standardabweichung werden benötigt, um zu zeigen, wie sehr die gefundenen Lösungen schwanken. Damit kann man eine Aussage treffen, ob der Algorithmus allein nur durch seine Natur (Zufallsprinzip) manchmal gute Ergebnisse liefern kann. Ziel dieser Arbeit ist es, den besten Median und geringste Standardabweichung für verschiedene Problemarten zu liefern. Das Minimum dient, um den besten Algorithmus zu bestimmen.

6.3.1 Cassini 1

Tabelle 6.1 – Ergebnisse aus 1000 Durchläufen für Cassini 1

Algorithmus	Minimum	Maximum	Median	St.Abw.
PASS	4,981	5,486	5,325	0,089
SPSO2011	5,266	19,903	11,669	3,344
DE	4,990	16.712	10,996	3,362
ABC	5,386	11,361	6,578	0,761
SA	4,994	36,797	12,975	4,351
CMA-ES	5,303	265,010	88,316	62,433

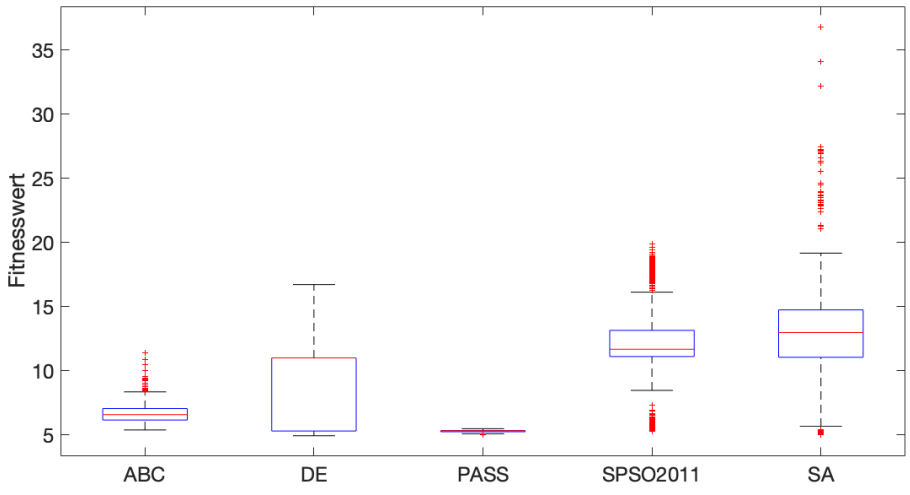
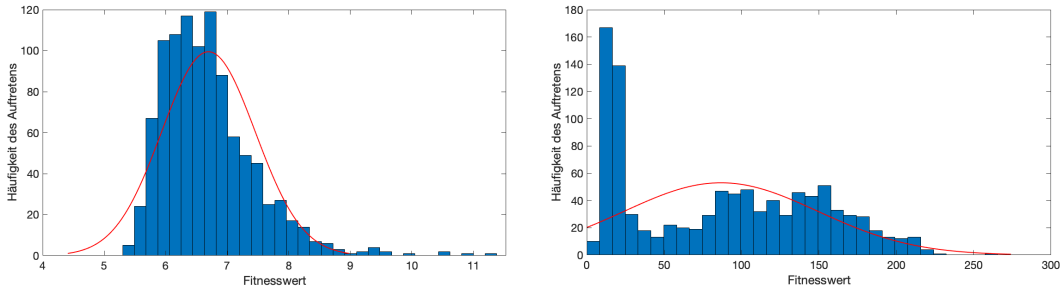
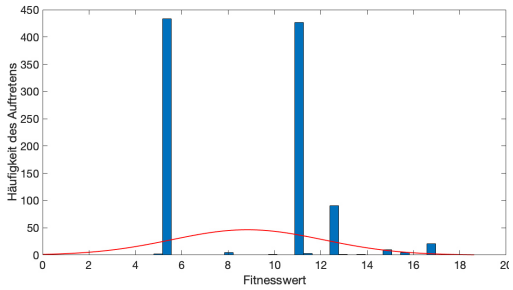


Abbildung 6.1 – Box-Plot für das Cassini 1 Problem. Die Ergebnisse von CMA-ES sind deutlich schlechter als die anderen und werden für den Vergleich nicht dargestellt.

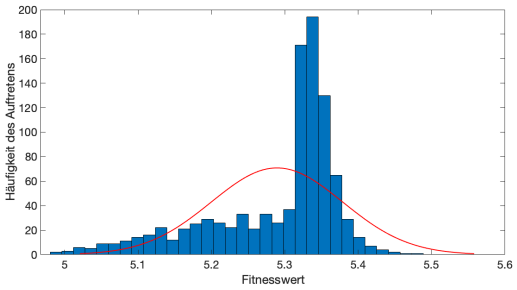


(a) ABC

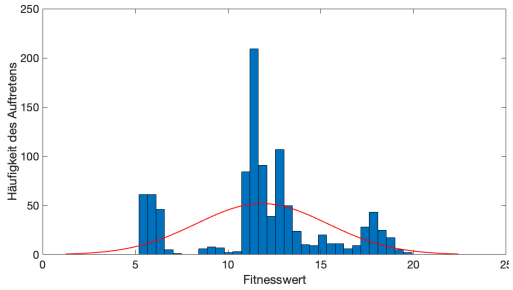
(b) CMA-ES



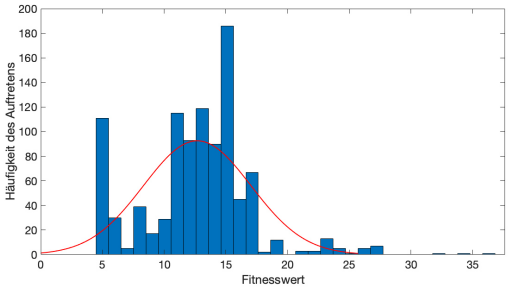
(c) DE



(d) PASS



(e) SPSO2011



(f) SA

Abbildung 6.2 – Die Histogramme der Algorithmen für das Cassini 1 Problem

6.3.2 GTOC 1

Tabelle 6.2 – Ergebnisse aus 1000 Durchläufen für GTOC 1

Algorithmus	Minimum	Maximum	Median	St.Abw.
PASS	-1,419 e+06	-7,527 e+05	-1,090 e+06	1,027 e+05
SPSO2011	-1,315 e+06	-3,983 e+04	-7,836 e+05	2,122 e+05
DE	-1,560 e+06	-5,503 e+04	-9,423 e+05	1,977 e+05
ABC	-9,343 e+05	-1,511 e+05	-4,743 e+05	1,399 e+05
SA	-1,116 e+06	-1,435 e+03	-4,720 e+04	2,056 e+05
CMA-ES	-1,230 e+06	-3,530 e-08	-6,931 e+01	1,605 e+05

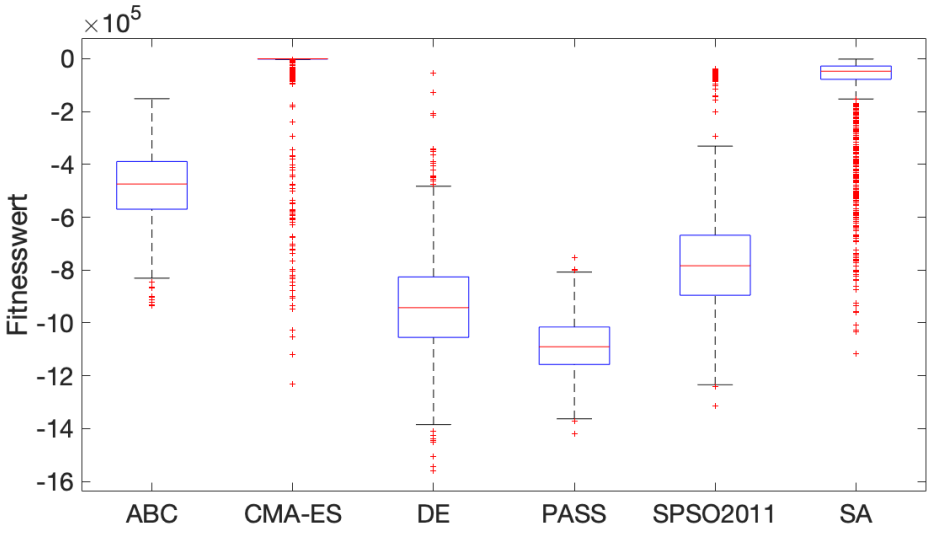
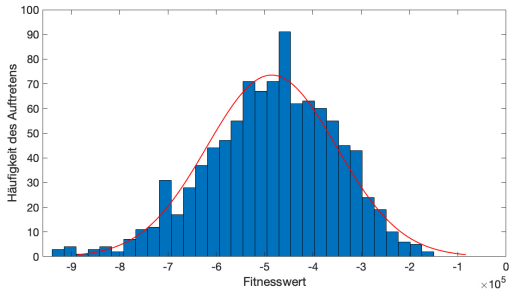
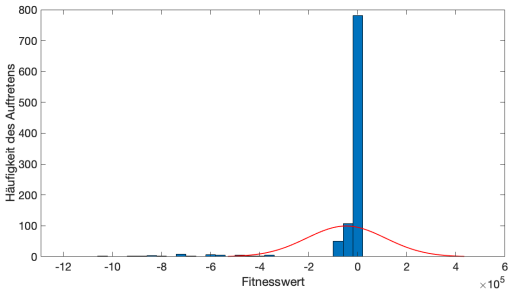


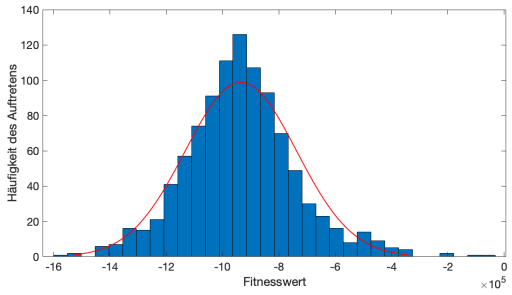
Abbildung 6.3 – Box-Plot für das GTOC 1 Problem



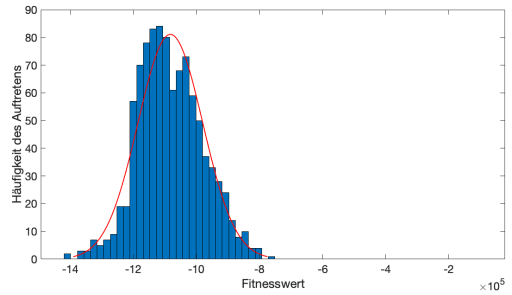
(a) ABC



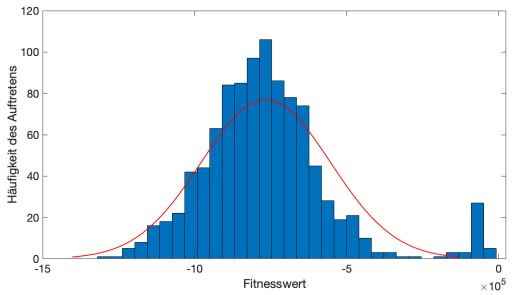
(b) CMA-ES



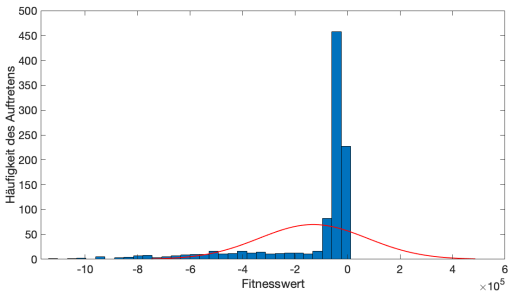
(c) DE



(d) PASS



(e) SPSO2011



(f) SA

Abbildung 6.4 – Die Histogramme der Algorithmen für das GTOC 1 Problem

6.3.3 Messenger

Tabelle 6.3 – Ergebnisse aus 1000 Durchläufen für Messenger

Algorithmus	Minimum	Maximum	Median	St.Abw.
PASS	8,357	19,501	14,462	1,234
SPSO2011	12,342	28,205	18,447	2,877
DE	7,523	21,070	16,823	2,003
ABC	10,660	32,385	22,580	3,542
SA	6,147	31,623	14,574	4,403
CMA-ES	47,999	265,922	130,744	35,132

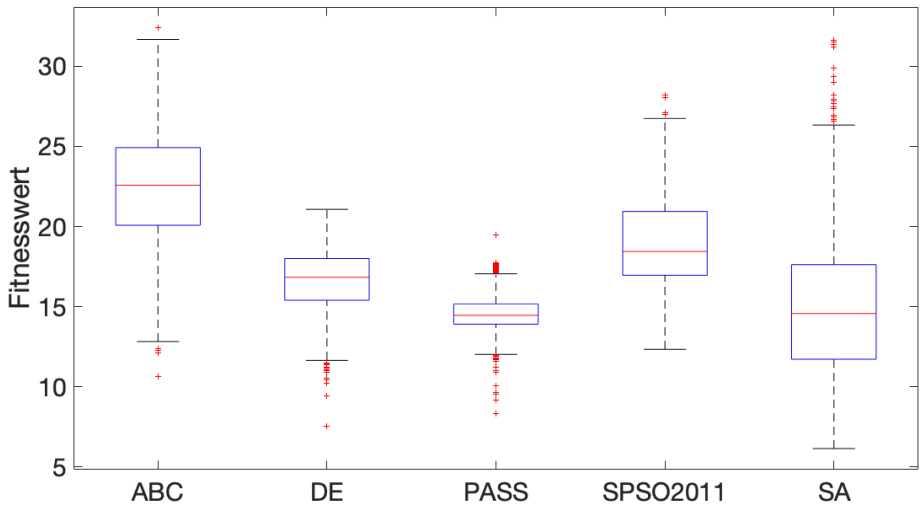
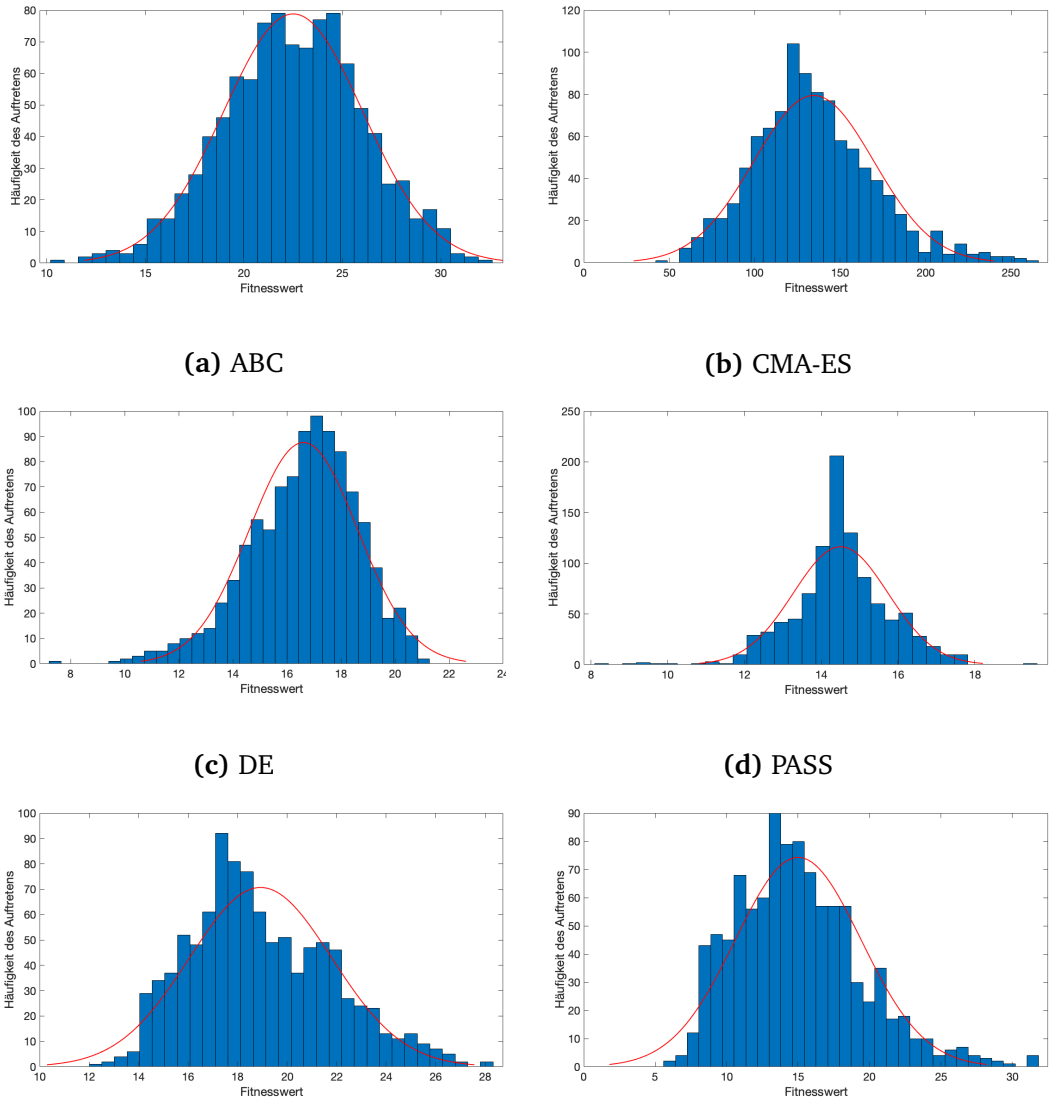


Abbildung 6.5 – Box-Plot für das Messenger Problem. Die Ergebnisse von CMA-ES sind deutlich schlechter als die anderen und werden für den Vergleich nicht dargestellt.



(a) ABC

(b) CMA-ES

(c) DE

(d) PASS

(e) SPSO2011

(f) SA

Abbildung 6.6 – Die Histogramme der Algorithmen für das Messenger Problem

6.3.4 Rosetta

Tabelle 6.4 – Ergebnisse aus 1000 Durchläufen für Rosetta

Algorithmus	Minimum	Maximum	Median	St.Abw.
PASS	1,364	6,892	2,702	0,743
SPSO2011	2,356	18,664	11,669	9,398
DE	1,994	11,497	5,894	1,976
ABC	3,902	18,548	11,247	2,382
SA	1,820	15,937	5,220	2,854
CMA-ES	39,970	201,971	71,945	18,723

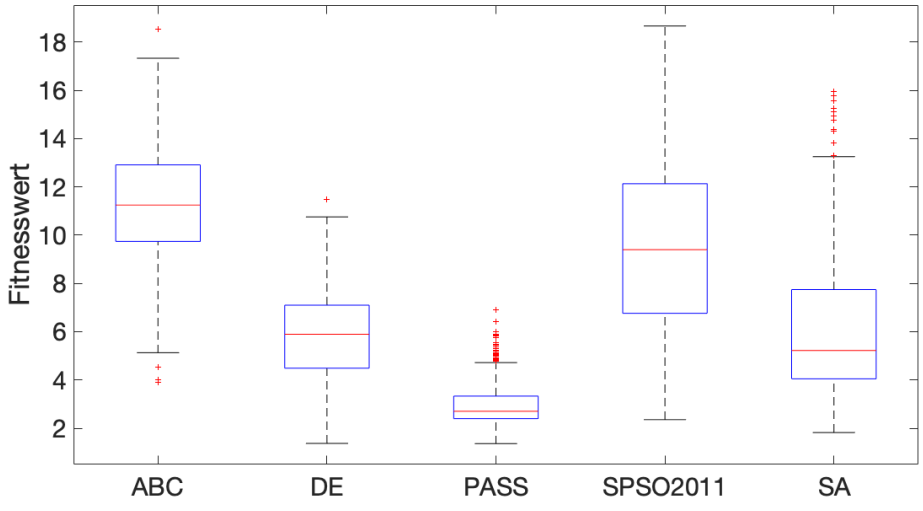
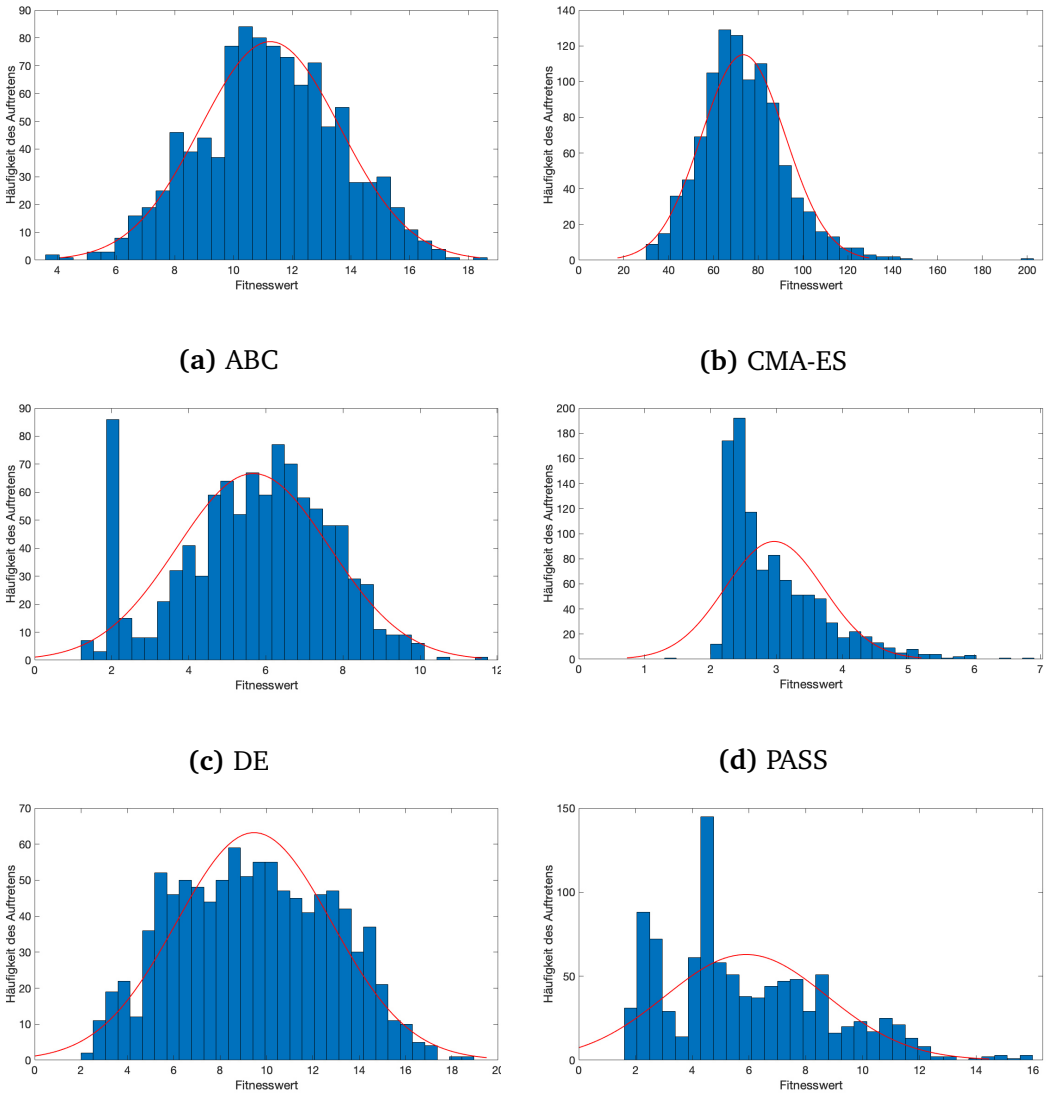


Abbildung 6.7 – Box-Plot für das Rosetta Problem. Die Ergebnisse von CMA-ES sind deutlich schlechter als die anderen und werden für den Vergleich nicht dargestellt.



(a) ABC **(b)** CMA-ES

(c) DE **(d)** PASS

(e) SPSO2011 **(f)** SA

Abbildung 6.8 – Die Histogramme der Algorithmen für das Rosetta Problem

6.4 Zusammenfassung

Es wurden alle vier Space Mission Probleme mit den vorgestellten Algorithmen evaluiert.

Für das Cassini 1 Problem liefert PASS im Durchschnitt die besten Ergebnisse. Das beste gefundene Minimum lautet **4,981** und wurde mit PASS gefunden (s. Tab. 6.1). Die anderen Algorithmen liefern sehr ähnliche Werte. Betrachtet man das Maximum und die Standardabweichung, dann sind die Schwankungen bei den anderen Algorithmen (außer ABC) im Vergleich zu PASS enorm. Am größten ist die Standardabweichung beim CMA-ES. Cassini 1 ist das einfachste Problem dieser Gruppe und die Ergebnisse, welche von PASS geliefert werden, liegen vergleichsweise in der Nähe des Minimums (s. Abb. 6.2).

Für das GTOC 1 Problem liefert PASS im Durchschnitt die besten Ergebnisse. Für das GTOC 1 Problem lautet das beste gefundene Minimum **-1,560 e+06** und wurde mit DE gefunden (s. Tab. 6.2). Die anderen Algorithmen liefern sehr ähnliche Werte. Betrachtet man das Maximum und die Standardabweichung, dann sind die Schwankungen bei den anderen Algorithmen im Vergleich zu PASS größer. Am größten ist die Standardabweichung beim CMA-ES. DE hat den besten Wert zwar gefunden, aber die Ergebnisse von PASS haben eine kleinere Standardabweichung und liegen sehr nah am besten gefundenen Wert. (s. Abb. 6.4).

Für das Messenger Problem liefert PASS im Durchschnitt die besten Ergebnisse. Für dieses Problem hat SA das beste Minimum mit **6,147** gefunden (s. Tab. 6.3). Die meisten Algorithmen liefern sehr ähnliche Werte. Nur CMA-ES und SPSO2011 können nicht so gut konvergieren. Betrachtet man das Maximum und die Standardabweichung, dann sind die Schwankungen bei den anderen Algorithmen im Vergleich zu PASS größer. Am größten sind die Schwankungen beim CMA-ES. Das Messenger Problem ist das schwierigste Problem, aber auch hier liefert PASS konstant gute Werte. (s. Abb. 6.6).

Für das Rosetta Problem liefert PASS im Durchschnitt die besten Ergebnisse. Wie bei Cassini 1 wurde auch hier das beste Minimum mit PASS gefunden und lautet **1,364** (s. Tab. 6.4). Die anderen Algorithmen (außer CMA-ES) liefern sehr ähnliche Werte. Betrachtet man das Maximum und die Standardabweichung, dann sind die Schwankungen bei den anderen Algorithmen im Vergleich zu PASS größer.

Am größten ist die Standardabweichung beim CMA-ES. Genau bei diesem Problem ist sehr gut zu sehen, dass PASS bessere Ergebnisse liefert (s. Abb. 6.8).

Die Ergebnisse der Evaluation zeigen, dass PASS im Durchschnitt die besten Werte liefert. CMA-ES liefert für die synthetischen BBPs gute Ergebnisse, aber für die schwierigen Space Mission Probleme sind die Ergebnisse deutlich schlechter als die der anderen Algorithmen. Das hat mit der Natur des Algorithmus zu tun. CMA-ES versucht, durch verschiedene Methoden Eigenschaften der Probleme zu extrahieren und sich diese zu Nutze zu machen [68]. Da die Space Probleme sehr komplex sind, klappt das nicht immer, und deshalb sind die Schwankungen so groß. Die besten gefundenen Werte für die Probleme wurden zweimal von PASS (Rosetta und Cassini 1), einmal von DE (GTOC 1) und einmal von SA (Messenger) geliefert.

Im nächsten Kapitel werden die Ergebnisse genauer diskutiert. Unter anderem wird diskutiert, welche Vorteile die vorgestellten Methoden und ihre Implementierungen liefern und wie die Ergebnisse zu interpretieren sind. Zum Schluss werden noch ein paar Punkte diskutiert, welche zur Verbesserung der Methodik beitragen können.

7

Fazit

Zusammenfassung und Ausblick

> Das Problem zu erkennen, ist wichtiger als die Lösung zu erkennen, denn die genaue Darstellung des Problems führt zur Lösung.
>
> ALBERT EINSTEIN

7.1 Zusammenfassung

In dieser Arbeit wurde das Framework PASS zur Unterstützung des Entwicklers bei der Lösung von BBPs vorgestellt. Die Architektur des Frameworks wurde in Kapitel 4 erläutert. Die dahinter liegende Methodik mit den einzelnen Komponenten wurde in Kapitel 5 im Detail erklärt und validiert. Im Kapitel 6 wurden die Methodik und ihre Implementierung mit anderen Algorithmen anhand der Space Mission Probleme evaluiert und verglichen.

Der Fokus dieser Arbeit lag auf dem Lösen von BBPs. Wie in Kapitel 1 erwähnt, gibt es drei Hauptaspekte, welche die Effizienz der Lösung durch die existierenden Methoden beeinträchtigen:

1. **Parallelisierung:** Die realen Probleme werden immer größer und komplizierter oder sie müssen in sehr kurzer Zeit gelöst werden, was erhebliche Ressourcen in Zeit und Hardware erfordert.

2. **Stillstand:** Optimierungsprobleme sind durch mehrere lokale Optima charakterisiert, die ein Verfahren zur Vermeidung einer zu frühen Konvergenz erfordern [24].

3. **Adaptivität:** Algorithmen erfordern einige problembedingte Anpassungen ihrer Verhaltensparameter, um bessere Ergebnisse zu erzielen [162].

Das Problem der **Parallelisierung** wurde durch die Umwandlung des Algorithmus von seriell zu parallel gelöst. Eine manuelle parallele Implementierung des PSO führt dazu, dass eine Parallelisierbarkeit von 90% bis 95% erreicht wird (s. Abb. 5.3). Die Parallelisierung wurde mittels OpenMP realisiert und die vorgestellte Methode der Hardwareanalyse funktioniert einwandfrei. Sie ist in der Lage, zu untersuchen, ob eine Parallelisierung für das zu lösende Problem überhaupt sinnvoll ist, und nach der Überprüfung wird die parallele oder die serielle Version des Algorithmus aktiviert. Für die schwierigen und komplizierten Probleme wird die parallele Version benutzt und für die Einfachen die serielle Version. Das Schalten zwischen den zwei Versionen erfolgt automatisiert und ohne Änderung im Quellcode.

Das Problem des **Stillstands** wurde durch das Benutzen des *Island Models* behandelt. Die Implementierung dieses Models wurde parallel durch MPI ermöglicht. Wie in Kapitel 5 durch die Validierung gezeigt, hilft diese Methode, meist die lokalen

Minima zu verlassen. Die parallele Implementierung dieses Modells ermöglicht auch eine Zeitersparnis und führt dazu, dass die Rechnerkapazität voll ausgenutzt wird.

Das Problem der **Adaptivität** wurde durch eine neue Methode behandelt. Die schon existierenden Methoden (s. Kap. 3) dauern zu lange und auf Basis der Literatur wurde ein neuer Algorithmus entwickelt, welcher die Suche deutlich verkürzt. Die gefundenen Werte für die Parameter führen zu einer besseren Anpassung des Algorithmus an das zu lösende Problem und somit zu besseren Ergebnissen (s. Kap 5).

Alle drei Hauptaspekte wurden in dem Framework PASS implementiert. Vieles läuft automatisiert und keine Änderungen im Quellcode sind notwendig. Das Bilden eines Kostenmodells der Rechnerarchitektur für die Bewertung der Parallelität und das Finden guter Parameter für den Algorithmus und deren Speicherung für die spätere Nutzung erfolgen automatisiert.

Das Ziel dieser Arbeit war das Liefern eines Algorithmus, welcher im **Durchschnitt** besser als die anderen bekannten Verfahren abschneidet. Betrachtet man die Ergebnisse in Kapitel 6, ist sehr gut zu sehen, dass PASS die besten Durchschnittswerte für den verwendeten Benchmark liefert. In vier Fällen hat PASS auch die absoluten minimalen Werte gefunden. Zweimal haben andere Algorithmen den besten Wert gefunden. Das war auch zu erwarten. Einen Algorithmus zu finden, der immer bessere Ergebnisse als alle anderen liefert, ist aufgrund der Erläuterung in Kapitel 3 (s. Def. 3.5.1: No Free Lunch Theorem) nicht möglich. Betrachtet man zusätzlich die Standardabweichung der Ergebnisse in Kapitel 6, ist auch gut zu sehen, dass die Werte von PASS eine sehr kleine Abweichung haben. Das zeigt, dass die gefundenen Werte über die 1000 Evaluationen des Problems immer gut sind und das Framework stets gute Ergebnisse liefert.

Den Schwerpunkt der untersuchten Algorithmen bildet der Particle Swarm Optimization (PSO) Algorithmus. Dieser Algorithmus wurde exemplarisch ausgewählt. Alles wurde softwaretechnisch durch das Framework PASS implementiert, welches in dem folgenden Abschnitt kurz vorgestellt wird.

Abbildung 7.1 – Das Framework-Logo

7.2 Die Software

Um die Methode und deren Auswirkung zu testen, wurde ein Framework entwickelt. Das PASS Framwork steht über Github[7] zur Verfügung und wird mit einer MIT-Lizenz[8] geliefert. Als Programmiersprachen wurden mit C++11 und C++14 moderne und sehr verbreitete Sprachen ausgewählt. Die Stärke der ganzen Methodik liegt auch in der Programmierung. Beide parallele Paradigmen MPI und OpenMP werden unterstützt und keine Codeänderung ist notwendig, um diese zu aktivieren oder zu deaktivieren. Weiterhin wird eine Unterstützung für SIMD Vektorisierung angeboten. Die Software skaliert mit der Hardware und funktioniert sowohl in Desktop PCs als auch auf einem HPCC. Es gibt nur eine Abhängigkeit von fremder Software und das ist Armadillo [127]. Die Installation und die Benutzung sind ebenfalls sehr einfach und dazu steht eine Dokumentation zur Verfügung[9]. Im Framework sind schon einige Probleme implementiert, welche für die Auswertung benutzt werden können. Die Implementierten BBPs sind die in dieser Arbeit betrachteten Probleme:

1. Black-box Benchmarking

   (a) Ackley Function

   (b) De Jong's Function

[7] https://github.com/rshuka/PASS
[8] https://github.com/rshuka/PASS/blob/master/LICENSE
[9] https://rshuka.github.io/PASSDoc/index.html

(c) Griewank Function

(d) Rastrigin Function

(e) Rosenbrock Function

(f) Schwefel Function

(g) Styblinski-Tang Function

(h) Sum of Different Powers Function

2. Real World Benchmarking

(a) GTOC 1

(b) Messenger (Full Version)

(c) Cassini 1

(d) Rosetta

Das Framework bietet auch die benötigten abstrakten Klassen, um es mit weiteren Algorithmen zu erweitern. Eine ausführliche API des Frameworks ist in der Dokumentation zu finden. Insgesamt sind es **4.905** Zeilen Code und **1.551** Zeilen Kommentare.

7.3 Zukünftige Forschungsziele

Im Laufe dieser Arbeit haben sich neue Forschungsmöglichkeiten für zukünftige Arbeiten ergeben. Diese sind:

- *Optimierung der Hardwareanalyse:* Für die Hardwareanalyse werden synthetische Probleme mit verschiedenen Laufzeiten als Trainingsdaten für das Modell erzeugt. Je höher diese Anzahl ist, desto länger dauert die Hardwarenanalyse und desto genauer ist das Modell. Momentan werden 120 Trainingsdaten erzeugt. Dieser Wert hat sich in den Tests auf dem SRA-Cluster und Desktop PCs als bester Wert herausgestellt. Diese Variable soll in anderen Architekturen und Rechnern weiter untersucht werden, um zu erfahren, ob es geeignete Werte gibt und ob es eine gewisse Abhängigkeit von der Größe des Wertes und der Architektur gibt.

- *Weitere Vorhersagemethoden untersuchen:* Für die Hardwareanalyse wurde eine Methode des maschinellen Lernens benutzt. Die benutzte Methode ist die polynomiale Regressionanalyse. Es sollen andere Methoden untersucht werden, um zu sehen, ob sie bessere Ergebnisse liefern oder ob sie besser für diesen Anwendungsfall geeignet wären.

- *Optimierung der Parametersuche:* Für das Finden der optimalen Parameter wurde ein neuer Algorithmus vorgestellt. Dieser Algorithmus hat zwei Variablen, welche vom Nutzer bestimmt werden. Die Granularität gibt an, wie genau der Parameter sein soll, und die Wiederholungsanzahl gibt an, wie oft die Optimierung wiederholt wird. Die Granularität ist auf 6 und die Wiederholungsanzahl ist auf 30 gesetzt. Je höher diese Werte sind, desto länger dauert die Suche. Es soll untersucht werden, ob irgendwelche Abhängigkeiten vorliegen und ob größere Werte tatsächlich auch bessere Ergebnisse liefern.

- *Optimierung des Island Models:* Für das *Island Model* wurde eine bestimmte Konfiguration benutzt (s. Abs. 5.3). Das Finden der besten Konfiguration für das *Island Model* ist sehr kompliziert und muss genau untersucht werden. Es müssen viele verschiedene Kommunikationsmodelle getestet und evaluiert werden, um zu schauen, welches am geeignetsten ist.

- *Zeitliche Optimierung des Island Models:* Wie schon im Abschnitt 5.3.4 erwähnt, kommt mit der MPI-Implementierung ein weiteres Problem dazu: Die Abhängigkeit von Dauer und Genauigkeit zwischen den Werten und der Anzahl der Schwärme bzw. benutzten Nodes. Je mehr Nodes benutzt werden, desto genauer sind die Werte und desto länger dauert die Suche. Wir haben es hier mit einer mehrkriteriellen Optimierung zu tun. Diese ist ein anderes Gebiet der Forschung in der Optimierungswelt. Die Konfiguration der Nodes ist auch sehr abhängig vom Ziel einer Evaluation. In der Domäne der Echtzeitsysteme ist z. B. die Zeit ein sehr wichtiger Faktor.

- *Mögliche Verbesserung der MPI Implementierung:* MPI bietet die Möglichkeit, eigene MPI-Befehle zu definieren und zu benutzen. Die aktuelle Lösung ist die schnellste Lösung durch existierende MPI-Befehle. Es soll untersucht werden, ob eine Lösung durch selbst definierte MPI-Befehle schneller ist.

- *Erweiterung des Frameworks:* Das Framework funktioniert für kontinuierliche Probleme. Einige Anpassungen würden ermöglichen, auch Probleme aus der Integer-Domäne zu lösen. Dadurch wären auch Probleme aus den Gebieten der Produktionsplanung, der Telekommunikationsnetze oder der Tourenplanung abgedeckt und die Methode kann auch für diese Domänen getestet werden.
- *Testen weiterer BBPs:* Es wäre interessant zu wissen, ob das Verfahren auch an weiteren realen Problemen aus verschiedenen Domänen (z. B. Wirtschaft) genauso gut funktioniert.
- *Testen weiterer Algorithmen:* Die vorgestellte Methodik sollte an anderen Algorithmen der im Abschnitt 3.7 definierten Algorithmengruppe getestet werden.

Abkürzungsverzeichnis

ACT	Advanced Concept Team
ASP	Algorithm Selection Problem
ACO	Ant Colony Optimization
API	Application Programming Interface
ABC	Artificial Bee Colony
BBOB	Black-Box Optimization Benchmark
BBP	Blackbox-Problem
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
DE	Differential Evolution
DRAM	Dynamic Random Access Memory
ESA	European Space Agency
EA	Evolutionäre Algorithmen
GPGPU	General Purpose Graphics Processing Unit
GP	Genetic Programming
GTOP	Global Trajectory Optimization

HPCC	High Performance Computing Cluster
HC	Hill Climbing
IWD	Intelligent Water Drop
LCS	Learning Classifier Systems
MPI	Message Passing Interface
NASA	National Aeronautics and Space Administration
NUMA	Non-Uniform Memory Access
openCL	Open Computing Language
OpenMP	Open Multi-Processing
PASS	Parallel Adaptive Swarm Search
PSO	Particle Swarm Optimization
SA	Simulated Annealing
SIMD	Single Instruction Multiple Data
SPSO2011	Standard Particle Swarm Optimization 2011
SHC	Stochastic Hill Climbing
SM	Streaming Multi-Processor
TS	Tabu Search
UMA	Uniform-Memory-Access

Abbildungsverzeichnis

Tabellenverzeichnis

Pseudocodeverzeichnis

Publikationen

[1] Romeo Shuka, Jürgen Brehm. *A Parallel Adaptive Swarm Search Framework for Solving Black-Box Optimization Problems.* In ARCS 2019; 32th International Conference on Architecture of Computing Systems 2019.

[2] Romeo Shuka, Sebastian Niemann, Jürgen Brehm, Christian Müller-Schloer. *Towards an Algorithm and Communication Cost Model for the Parallel Particle Swarm Optimization.* In ARCS 2016; 29th International Conference on Architecture of Computing Systems 2016.

[3] Romeo Shuka, Jürgen Brehm and Christian Müller-Schloer. *Adaptive Swarm Search.* In Poster presented at ARCS 2015; 28th International Conference on Architecture of Computing Systems 2015.

[4] Romeo Shuka. *Adaptive Swarm Search in Organic Computing.* Doctoral Dissertation Colloquium 2015, Kassel University Press GmbH 2015.

[5] Henner Heck, Sarah Edenhofer, Christian Gruhl, Andreas Lund, Romeo Shuka, Jörg Hähner. *On the Application Possibilities of Organic Computing Principles in Socio-technical Systems in Organic Computing.* Doctoral Dissertation Colloquium 2015, Kassel University Press GmbH 2015.

Literatur

[1] H. Abadlia, N. Smairi und K. Ghedira. „Particle Swarm Optimization Based on Dynamic Island Model“. In: *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, S. 709–716.

[2] Hazem Radwan Ahmed. „An Efficient Fitness-based Stagnation Detection Method for Particle Swarm Optimization“. In: *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*. GECCO Comp '14. New York, NY, USA: ACM, 2014, S. 1029–1032. ISBN: 978-1-4503-2881-4. DOI: 10.1145/2598394.2605669.

[3] Mahamad Alam, Biswarup Das und Vinay Pant. „A comparative study of metaheuristic optimization approaches for directional overcurrent relays coordination“. In: *Electric Power Systems Research* 128 (Nov. 2015), S. 39–52. DOI: 10.1016/j.epsr.2015.06.018.

[4] Abraham Adrian Albert. *Solid Analytic Geometry* -. Mineola, New York: Courier Dover Publications, 2016. ISBN: 978-0-486-81026-3.

[5] Okkes Tolga Altinoz und A. Egemen Yılmaz. „Comparison of Parallel CUDA and OpenMP Implementations of Particle Swarm Optimization“. In:

[6] Gene M. Amdahl. „Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities“. In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. AFIPS '67 (Spring). Atlantic City, New Jersey: ACM, 1967, S. 483–485. DOI: 10.1145/1465482.1465560.

[7] „Ant colonies for the travelling salesman problem“. In: *Biosystems* 43.2 (1997), S. 73 –81. ISSN: 0303-2647. DOI: https://doi.org/10.1016/S0303-2647(97)01708-5.

[8] The XXIIIrd International Astronomical Union General Assembly. *Resolution B1 on the Use of Julian Date*. 13. Feb. 2019. URL: https://www.iers.org/IERS/EN/Science/Recommendations/resolutionB1.html.

[9] Jagdish Bansal, Pramod Singh, Mukesh Saraswat, Abhishek Verma, Shimpi Jadon und Ajith Abraham. „Inertia Weight Strategies in Particle Swarm Optimization“. In: Okt. 2011, S. 633–640. DOI: 10.1109/NaBIC.2011.6089659.

[10] Marco Biazzini, Balázs Bánhelyi, Alberto Montresor und Márk Jelasity. „Distributed hyper-heuristics for real parameter optimization“. In: *GECCO*. ACM, 2009, S. 1339–1346.

[11] Christian Bierwirth. „Das Konzept der Fitnesslandschaft als Methode zur Beurteilung der Schwierigkeit von kombinatorischen Optimierungsprobleme“. In: *Intelligent Decision Support: Current Challenges and Approaches*. Berlin Heidelberg: Springer Science und Business Media, 2008. ISBN: 978-3-834-99777-7.

[12] Christian Blum und Andrea Roli. „Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison“. In: *ACM Comput. Surv.* 35.3 (Sep. 2003), S. 268–308. ISSN: 0360-0300. DOI: 10.1145/937503.937505.

[13] Eric Bonabeau, Marco Dorigo und Guy Theraulaz. *From Natural to Artificial Swarm Intelligence*. New York, NY, USA: Oxford University Press, Inc., 1999. ISBN: 0195131584.

[14] M. R. Bonyadi und Z. Michalewicz. „Analysis of Stability, Local Convergence, and Transformation Sensitivity of a Variant of the Particle Swarm Optimization Algorithm“. In: *IEEE Transactions on Evolutionary Computation* 20.3 (2016), S. 370–385. ISSN: 1089-778X. DOI: 10.1109/TEVC.2015.2460753.

[15] Mihaela Breaban, Lenuta Alboaie und Henri Luchian. „Guiding Users Within Trust Networks Using Swarm Algorithms“. In: *Proceedings of the Eleventh Conference on Congress on Evolutionary Computation*. CEC'09. Trondheim, Norway: IEEE Press, 2009, S. 1770–1777. ISBN: 978-1-4244-2958-5. URL: http://dl.acm.org/citation.cfm?id=1689599.1689832.

[16] Jason Brownlee. *Clever Algorithms: Nature-Inspired Programming Recipes*. 1st. Lulu.com, 2011. ISBN: 1446785068, 9781446785065.

[17] Emre Cakar. „Population-Based Runtime Optimisation in Static and Dynamic Environments“. PhD dissertation. Gottfried Wilhelm Leibniz Universität Hannover, 2011.

[18] Erick Cantú-Paz. *A Summary of Research on Parallel Genetic Algorithms*. 1995.

[19] Andrea Caponio, Ferrante Neri und Ville Tirronen. „Super-fit control adaptation in memetic differential evolution frameworks“. In: *Soft Computing* 13.8 (2008), S. 811. ISSN: 1433-7479. DOI: 10.1007/s00500-008-0357-1.

[20] C. S. Chang und Chung Min Kwan. „Evaluation of Evolutionary Algorithms for Multi-objective Train Schedule Optimization", booktitle=AI 2004: Advances in Artificial Intelligence, year=2005, publisher=Springer Berlin Heidelberg, address=Berlin, Heidelberg“. In: Hrsg. von Geoffrey I. Webb und Xinghuo Yu, S. 803–815. ISBN: 978-3-540-30549-1.

[21] Shu-Chuan Chu und Jeng-Shyang Pan. „Intelligent Parallel Particle Swarm Optimization Algorithms“. In: *Parallel Evolutionary Computations*. Hrsg. von Nadia Nedjah, Luiza de Macedo Mourelle und Enrique Alba. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, S. 159–175. ISBN: 978-3-540-32839-1. DOI: 10.1007/3-540-32839-4_8.

[22] M. Clerc und J. Kennedy. „The particle swarm - explosion, stability, and convergence in a multidimensional complex space“. In: *IEEE Transactions on Evolutionary Computation* 6.1 (2002), S. 58–73. ISSN: 1089-778X. DOI: 10.1109/4235.985692.

[23] Maurice Clerc. „Back to random topology“. unpublished. 2007.

[24] Maurice Clerc. „Stagnation Analysis in Particle Swarm Optimisation or What Happens When Nothing Happens“. 17 pages. Dez. 2006. URL: https://hal.archives-ouvertes.fr/hal-00122031.

[25] Maurice Clerc. „Standard Particle Swarm Optimisation“. 15 pages. Sep. 2012. URL: https://hal.archives-ouvertes.fr/hal-00764996.

[26] GNU Compiler. *GCC, the GNU Compiler Collection*. 13. Feb. 2019. URL: https://gcc.gnu.org/.

[27] *Compute Unified Device Architecture Website*. URL: https://www.nvidia.de/object/cuda-parallel-computing-de.html (besucht am 08. 01. 2019).

[28] Adaptive Computing. *TORQUE Resource Manager*. 13. Feb. 2019. URL: http://www.adaptivecomputing.com/products/torque/.

[29] Angelo Corana, Michele Marchesi, Claudio Martini und Sandro Ridella. „Corrigenda: "Minimizing Multimodal Functions of Continuous Variables with the 'Simulated Annealing' Algorithm"". In: *ACM Transactions on Mathematical Software - TOMS* 15 (Sep. 1989), S. 287. DOI: 10.1145/66888.356281.

[30] Pablo Cortés, Luis Onieva, Jesús Muáuzuri und Jose Guadix. „A Revision of Evolutionary Computation Techniques in Telecommunications and An Application for The Network Global Planning Problem". In: *Success in Evolutionary Computation*. Hrsg. von Ang Yang, Yin Shan und Lam Thu Bui. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, S. 239–262. ISBN: 978-3-540-76286-7. DOI: 10.1007/978-3-540-76286-7_11.

[31] Djurdje Cvijović und Jacek Klinowski. „Taboo Search: An Approach to the Multiple Minima Problem". In: *Science* 267.5198 (1995), S. 664–666. ISSN: 0036-8075. DOI: 10.1126/science.267.5198.664.

[32] Zbigniew J. Czech und Piotr Czarnas. „Parallel Simulated Annealing for the Vehicle Routing Problem with Time Windows". In: *Proceedings of the 10th Euromicro Conference on Parallel, Distributed and Network-based Processing*. EUROMICRO-PDP'02. Canary Islands, Spain: IEEE Computer Society, 2002, S. 376–383. ISBN: 0-7695-1444-8, 978-0-7695-1444-4.

[33] Marcin Molga Czeslav Smutnicki. *Test functions for optimization needs*. 13. Feb. 2019. URL: http://new.zsd.iiar.pwr.wroc.pl/files/docs/functions.pdf.

[34] L. Dagum und R. Menon. „OpenMP: an industry standard API for shared-memory programming". In: *IEEE Computational Science and Engineering* 5.1 (1998), S. 46–55. ISSN: 1070-9924. DOI: 10.1109/99.660313.

[35] S. S. Dahiya, J. K. Chhabra und S. Kumar. „Application of Artificial Bee Colony Algorithm to Software Testing". In: *2010 21st Australian Software Engineering Conference*. 2010, S. 149–154. DOI: 10.1109/ASWEC.2010.30.

[36] Narjess Dali und Sadok Bouamama. „GPU-PSO: Parallel Particle Swarm Optimization Approaches on Graphical Processing Unit for Constraint Reasoning: Case of Max-CSPs“. In: *Procedia Computer Science* 60 (2015), S. 1070 –1080. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.08.152.

[37] Charles Darwin. *On the origin of species*. New York :D. Appleton und Co., S. 470. URL: https://www.biodiversitylibrary.org/item/71804.

[38] Roozbeh Derakhshan, Bela Stantic, Othmar Korn und Frank K. H. A. Dehne. „Parallel Simulated Annealing for Materialized View Selection in Data Warehousing Environments“. In: *Algorithms and Architectures for Parallel Processing, 8th International Conference, ICA3PP 2008, Cyprus, June 9-11, 2008, Proceedings*. 2008, S. 121–132. DOI: 10.1007/978-3-540-69501-1_14.

[39] Bernard Derrida und Luca Peliti. „Evolution in a flat fitness landscape“. In: *Bulletin of Mathematical Biology* 53.3 (1991), S. 355–382. ISSN: 1522-9602. DOI: 10.1007/BF02460723.

[40] Peter Deuflhard und Andreas Hohmann. *Numerische Mathematik -*. 3. Aufl. Berlin: De Gruyter, 2002. ISBN: 978-3-110-17182-2.

[41] M. Dorigo, M. Birattari und T. Stutzle. „Ant colony optimization“. In: *IEEE Computational Intelligence Magazine* 1.4 (2006), S. 28–39. ISSN: 1556-603X. DOI: 10.1109/MCI.2006.329691.

[42] ESA. *Global Trajectory Optimisation Problems Database*. 13. Feb. 2019. URL: https://www.esa.int/gsp/ACT/projects/gtop/gtop.html.

[43] Eberhart und Yuhui Shi. „Particle swarm optimization: developments, applications and resources“. In: *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*. Bd. 1. 2001, 81–86 vol. 1. DOI: 10.1109/CEC.2001.934374.

[44] R. Eberhart und J. Kennedy. „A new optimizer using particle swarm theory“. In: *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. 1995, S. 39–43. DOI: 10.1109/MHS.1995.494215.

[45] Russ Eberhart, Pat Simpson und Roy Dobbins. *Computational Intelligence PC Tools*. San Diego, CA, USA: Academic Press Professional, Inc., 1996. ISBN: 0-12-228630-8.

[46] Matthias Ehrgott. *Multicriteria Optimization*. Berlin Heidelberg: Springer Science und Business Media, 2006. ISBN: 978-3-540-27659-3.

[47] Edwin J. Elton, Martin J. Gruber, Stephen J. Brown und William N. Goetzmann. *Modern Portfolio Theory and Investment Analysis. 9th Edition*. Wiley, 2014.

[48] Magnus Erik und Hvass Pedersen. „Good Parameters for Particle Swarm Optimization“. In: (Jan. 2010).

[49] Prof. Dr. Stephan Euler. *Netzwerk-Basiswissen, Teil 1*. 13. Feb. 2019. URL: https://www.tecchannel.de/a/netzwerk-basiswissen-teil-1,402420,5.

[50] G. I. Evers und M. Ben Ghalia. „Regrouping particle swarm optimization: A new global optimization algorithm with improved performance consistency across benchmarks“. In: *2009 IEEE International Conference on Systems, Man and Cybernetics*. 2009, S. 3901–3908. DOI: 10.1109/ICSMC.2009.5346625.

[51] Ludwig Fahrmeir, Thomas Kneib, Stefan Lang und Brian Marx. *Regression: Models, Methods and Applications*. Berlin: Springer-Verlag, 2013.

[52] C. A. Farrell und D. H. Kieronska. „Formal specification of SIMD execution“. In: *Proceedings of 1996 IEEE Second International Conference on Algorithms and Architectures for Parallel Processing, ICA/sup 3/PP '96*. 1996, S. 319–325. DOI: 10.1109/ICAPP.1996.562891.

[53] E. M. N. Figueiredo und T. B. Ludermir. „Effect of the PSO Topologies on the Performance of the PSO-ELM“. In: *2012 Brazilian Symposium on Neural Networks*. 2012, S. 178–183. DOI: 10.1109/SBRN.2012.26.

[54] *Fork-Join Model*. URL: https://commons.wikimedia.org/wiki/File:Fork_join.svg#filelinks (besucht am 08.01.2019).

[55] *GPU architecture hierarchy*. URL: http://www.prace-ri.eu/best-practice-guide-gpgpu-january-2017/ (besucht am 08.01.2019).

[56] Ahmed Hamdy Gad. „Space trajectories optimization using variable-chromosome-length genetic algorithms“. UMI 3474596. Diss. Michigan Technological University, 2011.

[57] Christian Gagné, Julie Beaulieu, Marc Parizeau und Simon Thibault. „Human-competitive lens system design with evolution strategies“. In: *Applied Soft Computing* 8.4 (2008). Soft Computing for Dynamic Data Mining, S. 1439 –1452. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2007.10.018.

[58] S. Geman und D. Geman. „Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-6.6 (1984), S. 721–741. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1984.4767596.

[59] Ashish Ghosh. *Evolutionary Computation in Data Mining* -. Berlin, Heidelberg: Springer, 2006. ISBN: 978-3-540-32358-7.

[60] Manfred Gilli und Peter Winker. *A Review of Heuristic Optimization Methods in Econometrics*. Swiss Finance Institute Research Paper No. 08-12, 2008.

[61] Manfred Gilli und Peter Winker. *Handbook of Computational Econometrics* -. New York: John Wiley und Sons, 2009. ISBN: 978-0-470-74890-9.

[62] Prometeus GmbH. *TOP 500 - The list*. 13. Feb. 2019. URL: https://www.top500.org/.

[63] E. B. Godshalk und D. H. Timothy. „Factor and principal component analyses as alternatives to index selection“. In: *Theoretical and Applied Genetics* 76.3 (1988), S. 352–360. ISSN: 1432-2242. DOI: 10.1007/BF00265334.

[64] Michael Gould. *GIScience grand challenges: How can research and technology in this field address big-picture problems?* 13. Feb. 2019. URL: https://www.esri.com/news/arcuser/1010/files/geochallenges.pdf.

[65] Prof. Dr.-Ing. Knut Graichen. „Methoden der Optimierung und optimalen Steuerung“. 2013.

[66] William Gropp, Ewing Lusk und Anthony Skjellum. „MPI - eine Einführung: portable parallele Programmierung mit dem Message-Passing Interface.“ In: (Jan. 2007).

[67] L. Hageman und D Young. *Applied Iterative Methods* -. Amsterdam: Elsevier, 2016. ISBN: 978-1-483-29437-7.

[68] N. Hansen, S. D. Müller und P. Koumoutsakos. „Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)“. In: *Evolutionary Computation* 11.1 (2003), S. 1–18. ISSN: 1063-6560. DOI: 10.1162/106365603321828970.

[69] Nikolaus Hansen. „The CMA Evolution Strategy: A Comparing Review“. In: *Towards a New Evolutionary Computation: Advances in the Estimation of Distribution Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, S. 75–102. ISBN: 978-3-540-32494-2. DOI: 10.1007/3-540-32494-1_4.

[70] Wim Hordijk. „Correlation Analysis of Coupled Fitness Landscapes“. In: *Recent Advances in the Theory and Application of Fitness Landscapes*. Hrsg. von Hendrik Richter und Andries Engelbrecht. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, S. 369–393. ISBN: 978-3-642-41888-4. DOI: 10.1007/978-3-642-41888-4_13.

[71] Shyh-Jier Huang und Xian-Zong Liu. „Application of artificial bee colony-based optimization for fault section estimation in power systems“. In: *International Journal of Electrical Power and Energy Systems* 44.1 (2013), S. 210 –218. ISSN: 0142-0615. DOI: https://doi.org/10.1016/j.ijepes.2012.07.012.

[72] Yueh-Min Huang, Yen-Ting Lin und Shu-Chen Cheng. „An Adaptive Testing System for Supporting Versatile Educational Assessment“. In: *Comput. Educ.* 52.1 (Jan. 2009), S. 53–67. ISSN: 0360-1315. DOI: 10.1016/j.compedu.2008.06.007.

[73] IEEE. *Congress on Evolutionary Computation*. 13. Feb. 2019. URL: http://cec2013.org/.

[74] Oscar Ibáñez, Lucia Ballerini, Oscar Cordón, Sergio Damas und José Santamaría. „An experimental study on the applicability of evolutionary algorithms to craniofacial superimposition in forensic identification“. In: *Information Sciences* 179.23 (2009), S. 3998 –4028. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2008.12.029.

[75] Lester Ingber und Bruce Rosen. „Genetic Algorithms and Very Fast Simulated Reannealing: A Comparison“. In: *Math. Comput. Model.* 16.11 (Nov. 1992), S. 87–100. ISSN: 0895-7177. DOI: 10.1016/0895-7177(92)90108-W.

[76] Mostafa Jamalipour, Reza Sayareh, Morteza Gharib, Farrokh Khoshahval und Mahmood Reza Karimi. „Quantum behaved Particle Swarm Optimization with Differential Mutation operator applied to WWER-1000 in-core fuel management optimization“. In: *Annals of Nuclear Energy* 54 (2013), S. 134–140. ISSN: 0306-4549. DOI: https://doi.org/10.1016/j.anucene.2012.11.008.

[77] Momin Jamil und Xin-She Yang. „A Literature Survey of Benchmark Functions For Global Optimization Problems“. In: *Int. J. of Mathematical Modelling and Numerical Optimisation* 4 (Aug. 2013). DOI: 10.1504/IJMMNO.2013.055204.

[78] Fei Jiang, Hugues Berry und Marc Schoenauer. „Unsupervised Learning of Echo State Networks: Balancing the Double Pole“. In: *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*. GECCO '08. Atlanta, GA, USA: ACM, 2008, S. 869–870. ISBN: 978-1-60558-130-9. DOI: 10.1145/1389095.1389264.

[79] K. Jong-Yul, J. Hee-Myung, L Hwa-Seok und P. June-Ho. „PC Cluster based Parallel PSO Algorithm for Optimal Power Flow“. In: *Proceedings of the International Conference on Intelligent Systems Applications to Power Systems*. 2007.

[80] Dieter Jungnickel. *Optimierungsmethoden - Eine Einführung*. 3. Aufl. Berlin Heidelberg New York: Springer-Verlag, 2014. ISBN: 978-3-642-54821-5.

[81] KITWARE. *Build with CMake. Build with Confidence.* 13. Feb. 2019. URL: https://cmake.org/.

[82] N. H. B. A. Kahar und A. F. Zobaa. „Optimal single tuned damped filter for mitigating harmonics using MIDACO“. In: *2017 IEEE International Conference on Environment and Electrical Engineering and 2017 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I CPS Europe)*. 2017, S. 1–4. DOI: 10.1109/EEEIC.2017.7977541.

[83] Dervis Karaboga und Bahriye Basturk. „A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm“. In: *Journal of Global Optimization* 39.3 (2007), S. 459–471. ISSN: 1573-2916. DOI: 10.1007/s10898-007-9149-x.

[84] J. Kaur und S. Singh. „Parallel Implementation of PSO Algorithm Using GPGPU“. In: *2016 Second International Conference on Computational Intelligence Communication Technology (CICT)*. 2016, S. 155–159. DOI: 10.1109/CICT.2016.38.

[85] Yousef S. Kavian, Arash Rashedi, Ali Mahani und Zabih Ghassemlooy. „Routing and wavelength assignment in optical networks using Artificial Bee Colony algorithm“. In: *Optik* 124.12 (2013), S. 1243 –1249. ISSN: 0030-4026. DOI: https://doi.org/10.1016/j.ijleo.2012.03.022.

[86] S. Kemble. *Interplanetary Mission Analysis and Design*. Springer Praxis Books. Springer Berlin Heidelberg, 2006. ISBN: 9783540376453. URL: https://books.google.de/books?id=m-k-qI6PsuUC.

[87] Wes Kendall. *MPI Tutorial*. 13. Feb. 2019. URL: http://mpitutorial.com/tutorials/.

[88] J. Kennedy und R. Eberhart. „Particle swarm optimization“. In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. Bd. 4. 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[89] David B. Kirk und Wen-Mei W Hwu. *Programming Massively Parallel Processors: A Hands-on Approach: Third Edition*. Elsevier Inc., Dez. 2016. ISBN: 9780128119860.

[90] S. Kirkpatrick, C. D. Gelatt und M. P. Vecchi. „Optimization by Simulated Annealing“. In: *Science* 220.4598 (1983), S. 671–680. ISSN: 0036-8075. DOI: 10.1126/science.220.4598.671.

[91] S.-H. Ko. „Sound Attenuation in Lined Rectangular Ducts with Flow and Its Application to the Reduction of Aircraft Engine Noise“. In: *Acoustical Society of America Journal* 50 (1971), S. 1418. DOI: 10.1121/1.1912784.

[92] Byung-Il Koh, Alan D. George, Raphael T. Haftka und Benjamin J. Fregly. „Parallel asynchronous particle swarm optimization“. In: *International Journal for Numerical Methods in Engineering* 67.4 (2006), S. 578–595. DOI: 10.1002/nme.1646.

[93] John R. Koza. „Genetic programming as a means for programming computers by natural selection“. In: *Statistics and Computing* 4.2 (1994), S. 87–112. ISSN: 1573-1375. DOI: 10.1007/BF00175355.

[94] Franz Kronthaler. *Statistik angewandt - Datenanalyse ist (k)eine Kunst mit dem R Commander*. 1. Aufl. 2016. Berlin Heidelberg New York: Springer-Verlag, 2015. ISBN: 978-3-662-47118-0.

[95] G. A. Laguna-Sánchez, M. Olguín-Carbajal, N. Cruz-Cortés, R. Barrón-Fernández und J. A. Álvarez Cedillo. „Comparative Study of Parallel Variants for a Particle Swarm Optimization Algorithm Implemented on a Multithreading GPU“. In: *Journal of Applied Research and Technology* (2009).

[96] Tim Kovac Larry Bull. *Foundations of Learning Classifier Systems* -. Berlin Heidelberg: Springer Science und Business Media, 2005. ISBN: 978-3-540-25073-9.

[97] B. D. H. Latter. „The Island Model of Poplation Differentiations: A general Solution“. In: *Genetics* 73.1 (1973), S. 147–157. ISSN: 0016-6731.

[98] Z. Liu, X. Li, W. Tan und Z. Zhang. „OpenMP-Based Multi-core Parallel Cooperative PSO with ICS Using Machine Learning for Global Optimization Problem“. In: *2015 IEEE International Conference on Systems, Man, and Cybernetics*. 2015.

[99] Thé Van Luong, Nouredine Melab und El-Ghazali Talbi. „GPU-based Island Model for Evolutionary Algorithms“. In: *Genetic and Evolutionary Computation Conference (GECCO)*. Portland, United States, 2010.

[100] Erik Magnus und Pedersen Hvass. „Good Parameters for Particle Swarm Optimization“. In: (Feb. 2010).

[101] James G. March. „Exploration and Exploitation in Organizational Learning“. In: *Organization Science* 2.1 (1991), S. 71–87. ISSN: 10477039, 15265455. URL: http://www.jstor.org/stable/2634940.

[102] Jürg T. Marti. „Konvexe Mengen in reellen Vektorräumen“. In: *Konvexe Analysis*. Basel: Birkhäuser Basel, 1977, S. 1–10. ISBN: 978-3-0348-5910-3. DOI: 10.1007/978-3-0348-5910-3_1.

[103] George Mavrotas. „Effective implementation of the ϵ-constraint method in Multi-Objective Mathematical Programming problems“. In: *Applied Mathematics and Computation* 213.2 (2009), S. 455 –465. ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2009.03.037.

[104] Andreas Meister. *Numerik linearer Gleichungssysteme - Eine Einführung in moderne Verfahren. Mit MATLAB®-Implementierungen von C. Vömel*. 5. Aufl. Berlin Heidelberg New York: Springer-Verlag, 2014. ISBN: 978-3-658-07200-1.

[105] Melllanox. *ConnectX®-2 VPI Single and Dual Port QSFP InfiniBand and Ethernet Adapter Card User Manual*. 13. Feb. 2019. URL: http://www.mellanox.com/related-docs/user_manuals/ConnectX%202_VPI_UserManual.pdf.

[106] R. Mendes, J. Kennedy und J. Neves. „The fully informed particle swarm: simpler, maybe better“. In: *IEEE Transactions on Evolutionary Computation* 8.3 (2004), S. 204–210. ISSN: 1089-778X. DOI: 10.1109/TEVC.2004.826074.

[107] Marjan Mernik, Shih-Hsi Liu, Dervis Karaboga und Matej Črepinšek. „On clarifying misconceptions when comparing variants of the Artificial Bee Colony Algorithm by offering a new implementation“. In: *Information Sciences* 291 (2015), S. 115 –127. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2014.08.040.

[108] *Message Passing Interface Website*. URL: https://www.mcs.anl.gov/research/projects/mpi/ (besucht am 08.01.2019).

[109] Robert M. Metcalfe und David R. Boggs. „Ethernet: Distributed Packet Switching for Local Computer Networks“. In: *Commun. ACM* 19.7 (Juli 1976), S. 395–404. ISSN: 0001-0782. DOI: 10.1145/360248.360253. URL: http://doi.acm.org/10.1145/360248.360253.

[110] Michalewicz, Zbigniew, Fogel und David B. *How to Solve It: Modern Heuristics* -. Berlin Heidelberg: Springer Science und Business Media, 2013. ISBN: 978-3-662-07807-5.

[111] Mario Andrés Muñoz Acosta, Yuan Sun, Michael Kirley und Saman Halgamuge. „Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges“. In: *Information Sciences* 317 (Mai 2015), S. 224–245. DOI: 10.1016/j.ins.2015.05.010.

[112] NASA. *Basics of Space Flight: Section 1: Environment, Chapter 4: Trajectories*. 13. Feb. 2019. URL: https://solarsystem.nasa.gov/basics/chapter4-1/.

[113] NASA. *Messenger Mission NASA*. 13. Feb. 2019. URL: https://www.nasa.gov/mission_pages/messenger/main/index.html#.U2hmY98chgI.

[114] J. Mukund Nilakantan und S. G. Ponnambalam. „An efficient PSO for type II robotic assembly line balancing problem“. In: *2012 IEEE International Conference on Automation Science and Engineering (CASE)*. 2012, S. 600–605. DOI: 10.1109/CoASE.2012.6386398.

[115] Yenny Noa Vargas und Stephen Chen. „Particle swarm optimization with resets - Improving the balance between exploration and exploitation“. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6438 LNAI.PART 2 (2010), S. 371–381. ISSN: 03029743. DOI: 10.1007/978-3-642-16773-7_32.

[116] *Open Computing Language Website*. URL: https://www.khronos.org/opencl/ (besucht am 08. 01. 2019).

[117] *Open Multi-Processing Website*. URL: https://www.openmp.org/ (besucht am 08. 01. 2019).

[118] *PGI Accelerator Compilers with OpenACC Directives*. URL: https://www.pgroup.com/resources/accel.htm (besucht am 08. 01. 2019).

[119] K. Pearson. „The Problem of the Random Walk“. In: *Nature* 72.1865 (1905), S. 294.

[120] David Corn Peter Bentley. *Creative Evolutionary Systems* -. 1950.

[121] G.F. Pfister. „An introduction to the infiniband architecture“. In: *High Performance Mass Storage and Parallel I/O* (Jan. 2001), S. 617–632.

[122] Oskar Pusz. „Entwurf interplanetarer Bahnkurven mittels Hint-basierter Optimierung“. Masterarbeit. Universität Hannover, 2016.

[123] *R-Stream® Automatic Polyhedral Optimizing Compiler*. URL: https://www.reservoir.com/product/r-stream/ (besucht am 08. 01. 2019).

[124] Erhard Rahm. „Mehrrechner-Datenbanksysteme“. In: *Synchronisation in Mehrrechner-Datenbanksystemen: Konzepte, Realisierungsformen und quantitative Bewertung*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, S. 6–19. ISBN: 978-3-642-74123-4. DOI: 10.1007/978-3-642-74123-4_2.

[125] V. Roberge und M. Tarbouchi. „Comparison of Parallel Particle Swarm Optimizers for Graphical Processing Units and Multicore Processors“. In: *Journal of Computational Intelligence and Applications* (2013).

[126] Stuart Russell und Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN: 0136042597, 9780136042594.

[127] Dr. Conrad Sanderson. *Armadillo: C++ library for linear algebra and scientific computing*. 13. Feb. 2019. URL: http://arma.sourceforge.net/.

[128] Steinbauer Schichl. *Einführung in das mathematische Arbeiten*. Berlin Heidelberg New York: Springer-Verlag, 2018. ISBN: 978-3-662-56806-4.

[129] Martin Schlueter. „MIDACO software performance on interplanetary trajectory benchmarks“. In: *Advances in Space Research* 54.4 (2014), S. 744 –754. ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2014.05.002.

[130] Berthold Immanuel Schmitt. „Konvergenzanalyse für die Partikelschwarmoptimierung“. In: *Ausgezeichnete Informatikdissertationen 2015*. Hrsg. von Steffen Hölldobler. Bonn: Gesellschaft für Informatik, 2015, S. 259–268.

[131] Jochen Schwarze. *Grundlagen der Statistik - Wahrscheinlichkeitsrechnung und induktive Statistik : [Lehrbuch]*. 9. vollständig überarbeitete Auflage. Herne: NWB, Verlag Neue Wirtschafts-Briefe, 2009. ISBN: 978-3-482-56869-5.

[132] Hamed Shah-Hosseini. „Intelligent water drops algorithm: A new optimization method for solving the multiple knapsack problem“. In: *International Journal of Intelligent Computing and Cybernetics* 1.2 (2008), S. 193–212. DOI: 10.1108/17563780810874717.

[133] Y. Shi und R. Eberhart. „A modified particle swarm optimizer“. In: *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*. 1998, S. 69–73. DOI: 10.1109/ICEC.1998.699146.

[134] Yuhui Shi und Russell C. Eberhart. „Parameter selection in particle swarm optimization“. In: *Evolutionary Programming VII*. Hrsg. von V. W. Porto, N. Saravanan, D. Waagen und A. E. Eiben. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, S. 591–600. ISBN: 978-3-540-68515-9.

[135] Oleg V. Shylo, Timothy Middelkoop und Panos M. Pardalos. „Restart Strategies in Optimization: Parallel and Serial Cases“. In: *Parallel Comput.* 37.1 (2011), S. 60–68. ISSN: 0167-8191. DOI: 10.1016/j.parco.2010.08.004.

[136] Xianhai Song, Hanming Gu, Li Tang, Sutao Zhao, Xueqiang Zhang, Lei Li und Jianquan Huang. „Application of artificial bee colony algorithm on surface wave data“. In: *Computers and Geosciences* 83 (2015), S. 219 –230. ISSN: 0098-3004. DOI: https://doi.org/10.1016/j.cageo.2015.07.010.

[137] *Spacecraft Trajectory Optimization*. Cambridge Aerospace Series. Cambridge University Press, 2010. DOI: 10.1017/CBO9780511778025.

[138] Maloth Srinivas und Lalit Patnaik. „Genetic Algorithms: A Survey“. In: *Computer* 27 (Juli 1994), S. 17 –26. DOI: 10.1109/2.294849.

[139] Rainer Storn und Kenneth Price. „Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces“. In: *Journal of Global Optimization* 11.4 (1997), S. 341–359. ISSN: 1573-2916. DOI: 10.1023/A:1008202821328.

[140] Giovanni Stracquadanio, Angelo La Ferla, Matteo De Felice und Giuseppe Nicosia. „Design of Robust Space Trajectories“. In: *SGAI Conf.* 2011.

[141] Christoph Strößner. „Particle Swarm Optimization and Parameter Selection“. unpublished. 2014.

[142] Sonja Surjanovic und Derek Bingham. *Optimization Test Problems*. 13. Feb. 2019. URL: https://www.sfu.ca/~ssurjano/optimization.html.

[143] Axel T Brünger, Paul Adams und Luke M Rice. „New applications of simulated annealing in X-ray crystallography and solution NMR“. In: *Structure (London, England : 1993)* 5 (Apr. 1997), S. 325–36. DOI: 10.1016/S0969-2126(97)00190-1.

[144] B.W. Hogg T. Ogilvie E. Swidenbank. „Use of data mining techniques in the performance monitoring and optimisation of a thermal power plant“. In: *IET Conference Proceedings* (1998), 7–7(1).

[145] T Takahashi und M Miura. „HMPP (Hybrid Multicore Parallel Programming) workbench“. In: 64 (Juni 2010), S. 814–818.

[146] Peyman Tavallali, Marianne Razavi und Sean Brady. „A non-linear data mining parameter selection algorithm for continuous variables“. In: *PLOS ONE* 12.11 (Nov. 2017), S. 1–26. DOI: 10.1371/journal.pone.0187676.

[147] Rajeev Thakur, Rolf Rabenseifner und William Gropp. „Optimization of collective communication operations in MPICH“. In: *International Journal of High Performance Computing Applications* 19.1 (2005), S. 49–66.

[148] The Ohio State University. *MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE*. 13. Feb. 2019. URL: http://mvapich.cse.ohio-state.edu/.

[149] Kaiming Wang, Yong Mao, Jiangtao Chen und Shiwei Yu. „The optimal research and development portfolio of low-carbon energy technologies: A study of China“. In: *Journal of Cleaner Production* 176 (2018), S. 1065–1077. ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2017.11.230.

[150] Thomas Weise. *Global Optimization Algorithms - Theory and Application*. Second. Online available at http://www.it-weise.de/. Self-Published. URL: http://www.it-weise.de/.

[151] Dirk Werner. *Funktionalanalysis* -. 7. Aufl. Berlin Heidelberg New York: Springer-Verlag, 2011. ISBN: 978-3-642-21017-4.

[152] Darrell Whitley, Soraya Rana und Robert Heckendorn. „The Island Model Genetic Algorithm: On Separability, Population Size and Convergence“. In: *Journal of Computing and Information Technology* 7 (Dez. 1998).

[153] David Wolpert und William Macready. „No Free Lunch Theorems for Search“. In: (März 1996).

[154] Chukiat Worasucheep. „A Particle Swarm Optimization with stagnation detection and dispersion“. In: *2008 IEEE Congress on Evolutionary Computation, CEC 2008* (2008), S. 424–429. DOI: 10.1109/CEC.2008.4630832.

[155] Sewall Wright. „The roles of mutation, inbreeding, crossbreeding and selection in evolution“. In: *Proceedings of the Sixth International Congress of Genetics* 1 (1932), S. 356–366.

[156] D. Wu und Hao Gao. „Study on asynchronous update mechanism in Particle Swarm Optimization“. In: *2014 14th International Symposium on Communications and Information Technologies (ISCIT)*. 2014, S. 90–93. DOI: 10.1109/ISCIT.2014.7011876.

[157] Hongmei Yan, Jun Zheng, Yingtao Jiang, Chenglin Peng und Shouzhong Xiao. „Selecting Critical Clinical Features for Heart Diseases Diagnosis with a Real-coded Genetic Algorithm“. In: *Appl. Soft Comput.* 8.2 (März 2008), S. 1105–1111. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2007.05.017.

[158] Shengxiang Yang und Changhe Li. „A Clustering Particle Swarm Optimizer for Locating and Tracking Multiple Optima in Dynamic Environments“. In: *IEEE Transactions on Evolutionary Computation* 14 (Dez. 2010), S. 959 –974. DOI: 10.1109/TEVC.2010.2046667.

[159] M. Zambrano-Bigiarini, M. Clerc und R. Rojas. „Standard Particle Swarm Optimisation 2011 at CEC-2013: A baseline for future PSO improvements“. In: *2013 IEEE Congress on Evolutionary Computation*. 2013, S. 2337–2344. DOI: 10.1109/CEC.2013.6557848.

[160] Yudong Zhang, Praveen Agarwal, Vishal Bhatnagar, Saeed Balochian und Xuewu Zhang. „Swarm intelligence and its applications 2014“. In: *TheScientificWorldJournal* 2014 (2014), S. 204294. ISSN: 1537-744X. DOI: 10.1155/2014/204294.

[161] Hanhong Zhu, Yi Wang, Kesheng Wang und Yun Chen. „Particle Swarm Optimization (PSO) for the constrained portfolio optimization problem“. In: *Expert Systems with Applications* 38.8 (2011), S. 10161 –10169. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2011.02.075.

[162] X. Zhu und P. Milanfar. „Automatic Parameter Selection for Denoising Algorithms Using a No-Reference Measure of Image Content“. In: *IEEE Transactions on Image Processing* 19.12 (2010), S. 3116–3132. ISSN: 1057-7149. DOI: 10.1109/TIP.2010.2052820.

[163] White gecko. *Der Speedup einer parallel bearbeiteten Operation auf bis zu 16 CPUs*. 13. Feb. 2019. URL: https://commons.wikimedia.org/wiki/File:Speedup.svg.

Lebenslauf

Name:	Romeo Shuka
Geburtsdatum:	20.10.1985
Geburtsort:	Durres, Albanien
Staatsangehörigkeit:	deutsch
Schulausbildung :	1997 - 2004 Gymnasium Naim Frasheri, Durres, Albanien Abschluss: Abitur
Bachelorstudium:	2007 - 2012 Bachelorstudium an der Fakultät für Elektrotechnik und Informatik der Leibniz Universität Hannover Abschluss: B.Sc. Informatik
Masterstudium:	2012 - 2014 Masterstudium an der Fakultät für Elektrotechnik und Informatik der Leibniz Universität Hannover Abschluss: M.Sc. Informatik
Berufstätigkeit:	2014 - 2019 wissenschaftlicher Mitarbeiter am Institut für Systems Engineering, Fachgebiet System- und Rechnerarchitektur an der Leibniz Universität Hannover